MASTER EN SCIENCES INFORMATIQUES

Les réseaux intelligents et Internet
monitoring d'un IN via le Web

Michiels, Benoît

Award date:
1998

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX,

NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

Les Réseaux Intelligents & Internet - Monitoring d'un IN via le Web -

Benoît MICHIELS

Mémoire présenté en vue de l'obtention du grade de

Licencié en Informatique

Année Académique 1997 - 1998

Résumé

Le mémoire se propose d'analyser les Réseaux Intelligents (*Intelligent Networks*) et certaines des caractéristiques d'Internet. Ces deux sujets sont envisagés à la fois sous l'angle du « pourquoi » et sous l'angle du « comment », puis ils sont mis en relation. Cette mise en relation se fait sous la forme de l'implémentation d'un prototype d'application de monitoring de Réseau Intelligent.

Abstract

The thesis aims to analyze Intelligent Networks and some of the characteristics of the Internet. These two topics are studied both in term of « why » and in term of « how », then their relations are examined. This is done by an implementation of an Intelligent Network monitoring application prototype.

Avant-propos

Dès à présent, nous souhaitons faire part de nos remerciements les plus vifs à toutes celles et tous ceux qui ont participé, de près ou de loin, à l'élaboration de ce mémoire, et en particulier :

- M. Philippe van Bastelaer, des Facultés universitaires Notre-Dame de la Paix à Namur, promoteur du mémoire ;
- MM. Denis Fisette, Benoît Quirynen et Michel Degroot, d'Alcatel Namur, pour leur accueil et l'aide qu'ils ont apportée à la définition du projet.

Nous profitons aussi de cet espace avant-propos pour mettre le lecteur au fait des conventions que nous avons utilisées lors de la rédaction.

Le style choisi pour la citation est l'italique, avec retrait à droite lorsqu'un paragraphe complet est cité en entier. Les notes bibliographiques se situent en bas de page, et une bibliographie, qui se trouve à la fin du mémoire, reprend l'ensemble des ouvrages qui nous ont aidé à réaliser ce travail. Elle est précédée d'une liste d'abréviations, précaution nécessaire pour les exposés ayant trait au monde des télécommunications ; elle est suivie par une partie annexe reprenant certains éléments que nous n'avons pas cru devoir insérer dans le corps du travail.

La mise en page a un but principal de clarté, c'est pourquoi la table des matières (à laquelle succède la table des figures), qui suit cet avant-propos, a été poussée jusqu'au dernier niveau de titre.[1]

Nous espérons que le lecteur prendra autant de plaisir à parcourir l'ouvrage que nous en avons eu à le rédiger, et nous lui souhaitons une agréable lecture.

Benoît Michiels

[1] Que le lecteur ne s'effraie pas du nombre de pages qui composent ce mémoire : si le corps du travail tient en une centaine de pages, quatre annexes relativement longues l'accompagnent, l'une offrant des exemples de services de Réseau Intelligent, la deuxième donnant certains compléments à l'exposé des protocoles analysés, la troisième étant le lieu du code de l'application développée, la quatrième montrant certaines captures d'écran de cette application en fonctionnement.

Table des matières

Résumé i

Abstract i

Avant-propos ii

Table des matières iii

Table des illustrations vii

Introduction 1

Partie I. Les Réseaux Intelligents (Intelligent Networks – IN) 3

0. Introduction 3

1. Les Réseaux Intelligents : pourquoi ? 4
1.1. Les Réseaux Intelligents : une brève définition 4
1.2. Les Acteurs 5
1.2.1. Les acteurs classiques 5
1.2.2. Les acteurs spécifiques 7
1.2.3. Les organismes de standardisation 8
1.2.4. Les fournisseurs 9
1.3. Contexte du développement des IN 10
1.4. Conclusion : les Réseaux Intelligents répondent à une attente 12

2. Les Réseaux Intelligents : comment ? 15
2.0. Introduction 15
2.1. Le cycle de vie du service 16
2.1.1. La phase de conception 17
2.1.2. La phase de production 18
2.1.3. La phase d'intégration 21
2.1.4. La phase de remplacement 21
2.2. L'architecture physique d'un Réseau Intelligent 22
2.2.1. SSP : Service Switching Point 23
2.2.2. SCP : Service Control Point 23
2.2.3. IP : Intelligent Peripheral 24
2.2.4. SMP : Service Management Point 25
2.2.5. SCE : Service Creation Environment (1) 26
2.3. L'architecture des services IN 26
2.3.1. La logique de service en temps-réel 27
2.3.2. Le bloc de gestion du service 27
2.3.3. SIB : Service-Independent Building Blocks 28
2.3.4. SCE : Service Creation Environment (2) 29
2.3.4.1. Développement de SIB 29
2.3.4.2. Développement de services 30
2.3.4.3. Adaptation (*customization*) de services 30

3. Conclusion de la première partie 31

Partie II. Internet & les Réseaux Intelligents 32

0. Introduction 32

1. Internet : pourquoi ? 33
1.1. Internet : une brève définition 33
1.2. Internet : un bref historique 34

1.2.1. Origine militaire 34
1.2.2. Relève universitaire 36
1.2.3. Poursuite du développement 36
1.3. Des tendances actuelles d'Internet 37
1.3.1. Le commerce électronique 37
1.3.2. La technologie Push 39
1.4. Conclusion : pourquoi s'intéresser à Internet dans le cadre des IN ? 42

2. Internet : comment ? 44
2.0. Introduction 44
2.1. Caractéristiques d'Internet 44
2.1.1. Principes généraux d'interconnexion 44
2.1.2. Interconnexion au niveau de la couche IP 46
2.2. Protocoles 48
2.2.1. IP 48
2.2.1.1. Introduction 48
2.2.1.2. Caractéristiques d'IP 48
2.2.1.3. Adressage IP 49
2.2.1.4. Primitives de service 49
2.2.1.5. Format du datagramme IP 49
2.2.2. TCP 50
2.2.2.1. Introduction 50
2.2.2.2. Caractéristiques de TCP 50
2.2.2.3. Identification des processus 52
2.2.2.4. Ouverture d'une connexion 52
a) Ouverture passive 52
b) Ouverture active 53
2.2.2.5. Transfert de données 54
a) Transfert de données standard 54
b) Mécanisme du transfert de données 55
c) Transfert de données urgentes 57
2.2.2.6. Primitives de service 57
2.2.2.7. Format du segment TCP 58
2.2.3. HTTP 58
2.2.3.1. Introduction 58
2.2.3.2. Caractéristiques de HTTP 59
2.2.3.3. Primitives de service 60
2.2.3.4. Format du PDU HTTP 60
2.3. Langages 60
2.3.1. HTML 60
2.3.1.1. Un langage issu de SGML 60
2.3.1.2. Caractéristiques et possibilités de HTML 62
a) HTML 1.0 62
b) HTML 2.0 63
c) HTML 3.0 63
d) Extensions HTML de Netscape 64
e) Extensions HTML de *Internet Explorer* 64
f) Extensions HTML pour Java 65
2.3.2. Java 65
2.3.2.1. Introduction : Java répond à un besoin 65
2.3.2.2. Caractéristiques générales de Java 66
a) Java est orienté objet 66
b) Java est distribué 67
c) Java est interprété 67
d) Java est typé 68
e) Java est sécurisé 68
f) Java est multithread 69

3. Conclusion de la deuxième partie 71

Partie III. Monitoring d'un IN via le Web 72

0. Introduction : définition du problème 72

1. Choix de solutions 73
1.1. Éventail de possibilités 73
1.1.1. Le Push 73
1.1.2. Le courrier électronique 74
1.2. Critique des possibilités 75
1.2.1. Le Push 75
1.2.2. Le courrier électronique 76
1.3. Conclusion 76

2. Implémentation 77
2.1. Approches successives 77
2.1.1. Première approche (large) 77
2.1.1.1. Description générale 77
2.1.1.2. Découpe en modules 78
2.1.1.3. Description des modules 80
a) Le module *Form* 80
b) Le module *Subscript* 81
c) Le module *Collect* 82
d) Le module *Send* 82
e) Le module *Listener* 82
2.1.2. Seconde approche (restreinte) 86
2.2. Prototype 87
2.2.0. Introduction 87
2.2.1. Développement d'un générateur d'événements 87
2.2.2. Ajout d'un filtre au générateur d'événements 89
2.2.3. Développement d'une architecture Client/Serveur 90
2.2.3.1. Le serveur 90
2.2.3.2. Le client 91
2.2.4. Intégration du générateur d'événements au serveur 91
2.2.4.1. Le serveur 91
2.2.4.2. Le client 92

3. Conclusion de la troisième partie 93

Conclusion générale 95

Abréviations 98

Bibliographie 99

1. Ouvrages 99

2. Ressources *on-line* 99

Annexes 100

1. Exemples 100
1.1. Exemples de services IN 100
1.1.1. Advanced freephone, Universal access number, Kiosk, Automatic call distribution 100
1.1.2. Credit card billing, Alternate billing, Prepaid card call 100
1.1.3. Personal number, Universal personal telecommunications 101
1.1.4. Virtual Private network 101
1.1.5. Vote and opinion poll, Mass calling 101
1.2. Création d'un service 101

2. Compléments sur les protocoles 103
2.1. Le protocole IP 103
2.1.1. Adressage IP 103

2.1.2. Primitives de service IP 104
2.1.3. Format du datagramme IP 105
2.2. Le protocole TCP 106
2.2.1. Primitives de service TCP 106
a) Primitives liées à la connexion TCP 107
1. Ouvertures passive et active 107
2. Libération 108
3. Etat 109
b) Primitives liées au transfert de données 109
1. Envoi et livraison 109
2. Gestion de la fenêtre et erreurs 110
c) Paramètres des primitives de service 110
2.2.2. Format du segment TCP 111
2.3. Le protocole HTTP 112
2.3.1. Primitives de service HTTP 112
a) Primitives côté client 113
1. Service offert par le client WWW 113
2. Service offert par le client HTTP 113
b) Primitives côté serveur 114
1. Service offert par le serveur HTTP 114
2. Service offert par le serveur WWW 114
2.3.2. Format du PDU HTTP 114

3. Code de l'implémentation 117
3.1. Le générateur d'événements 117
3.1.1. Première approche : la classe Generateur 117
3.1.2. Seconde approche : la classe Alarme 119
3.2. Le générateur d'événements muni d'un filtre 121
3.3. L'architecture Client/Serveur 125
3.3.1. Le serveur 125
3.3.2. Le client 127
3.4. Le Serveur et le Client définitifs 129
3.4.1. Le serveur 129
3.4.2. Le client 138
3.4.3. La classe Alarme 141

4. Captures d'écran 145
4.1. Le générateur d'événements 145
4.2. Le générateur d'événements muni d'un filtre 145
4.3. L'architecture Client/Serveur 146
4.3.1. Le serveur 146
4.3.2. Le client 146
4.4. Le serveur et le client définitifs 147
4.4.1. La phase d'identification et acquisition des paramètres 147
4.4.1.1. Le serveur 147
4.4.1.2. Le client 147
4.4.2. La fin de la phase de monitoring 148
4.4.2.1. Le serveur 148
4.4.2.2. Le client 148

Table des illustrations

Figure 1 : Acteurs « classiques » d'un IN 6
Figure 2 : Chaîne de création d'un service IN 20
Figure 3 : Architecture physique d'un IN 22
Figure 4 : Structure en couches du modèle TCP/IP 45
Figure 5 : Interconnexion TCP/IP 47
Figure 6 : Schéma d'un socket 52
Figure 7 : Ouverture d'une connexion TCP 54
Figure 8 : Mécanisme d'acquittement et fenêtre 56
Figure 9 : Architecture générale du WWW 59
Figure 10 : Architecture du système de monitoring (1e approche) 79
Figure 11 : Scénario de fonctionnement du monitoring (1e approche) 85
Figure 12 : Classes d'adresses IP 103
Figure 13 : Primitives de service IP 105
Figure 14 : Format du datagramme IP 105
Figure 15 : Primitives liées à la connexion TCP – ouvertures passive et active 108
Figure 16 : Primitives liées à la connexion TCP – libération 108
Figure 17 : Primitives liées à la connexion TCP – état 109
Figure 18 : Primitives liées au transfert de données - envoi et livraison 109
Figure 19 : Primitives liées au transfert de données - gestion de la fenêtre et erreurs 110
Figure 20 : Format du segment TCP 111
Figure 21 : Primitives côté client – service offert par le client WWW 113
Figure 22 : Primitives côté client – service offert par le client HTTP 113
Figure 23 : Primitives côté serveur – service offert par le serveur HTTP 114
Figure 24 : Primitives côté serveur – service offert par le serveur WWW 114
Figure 25 : Format d'un PDU HTTP de requête 115
Figure 26 : Format d'un PDU HTTP de réponse 116

Introduction

Le monde des télécommunications, tel qu'il se présente aujourd'hui, peut être considéré sous l'angle de deux de ses caractéristiques. La première est le concept de rapidité, la seconde est celui de service.

La notion de rapidité peut s'envisager elle-même selon une double perspective. La première est ce que l'on pourrait appeler la rapidité "matérielle". Elle concerne les évolutions technologiques qui permettent aux données de circuler toujours plus vite et en plus grand nombre, sur des supports toujours plus performants et souvent meilleur marché, donc accessibles au plus grand nombre. Elle est liée aux progrès réalisés à la fois au niveau des matériels et des performances des logiciels. La seconde est la rapidité des acteurs du monde des télécommunications qui, elle, n'est pas d'abord un progrès, mais une exigence dictée par les lois du marché. Elle est liée aux capacités de développement de matériels et de logiciels, en termes d'efficacité et d'adéquation à la demande, à la vitesse de réaction face à un environnement sans cesse changeant et à une vision claire des stratégies gagnantes à plus ou moins long terme.

A propos du service, l'on note une demande de plus en plus poussée de la part des utilisateurs, tant professionnels que particuliers, notamment au niveau des capacités et des vitesses de transmission des données, mais aussi au niveau de l'interactivité, de l'étendue et de la mise à disposition des possibilités offertes par les nouveaux moyens de communication.

Cet ensemble de caractéristiques se retrouve dans le monde des Réseaux Intelligents, et c'est ce que nous allons tâcher de faire transparaître tout au long des pages de ce mémoire. A cette fin, le travail est organisé comme suit. Le mémoire comporte trois parties qui, chacune, répondent à une double préoccupation : respectivement celle du pourquoi et celle du comment.

La première partie est un texte ayant pour objet les Réseaux Intelligents. Nous pouvons dès à présent donner une définition de ce type de réseau, afin de permettre au lecteur peut-être non familiarisé avec ce sujet de pouvoir entrer de plain-pied dans le corps du travail. Cette définition est celle donnée par Bell-Atlantic dans son *tutorial*[2] sur les IN :

> *An intelligent network (IN) is a service-independent telecommunications network. That is, intelligence is taken out of the switch and placed in computer nodes that are distributed throughout the network. This provides the network operator with the means to develop and control services more efficiently. New capabilities can be rapidly introduced into the network. Once introduced, services are easily customized to meet individual customer's needs.*

Après une autre brève définition sont donnés successivement des exposés relatifs aux acteurs qui interviennent dans ce monde, au cycle de vie des services IN et aux architectures des IN et des services qu'ils proposent.

La deuxième partie s'attache à la description de certaines des caractéristiques d'Internet, et plus particulièrement du *World-Wide Web*. Il s'agira de donner une définition, de retracer brièvement l'historique de la toile et d'examiner d'un peu près les réalités technologiques que le terme Internet recouvre, notamment en termes de protocoles et de langages informatiques.

La troisième partie se compose des notes relatives à l'implémentation d'un prototype d'application de monitoring de Réseau Intelligent. Les approches successives en sont données, ainsi que les étapes qui en ont jalonné la finalisation, et le code (en annexe).

---

[2] www.webforum.com/bell-atlantic. Nous conservons ici l'anglais afin de trahir le moins possible l'esprit du texte.

Partie I. Les Réseaux Intelligents (*Intelligent Networks – IN*)

0. Introduction

Comme nous l'avons annoncé dans l'introduction, le concept des Réseaux Intelligents sera abordé selon une double perspective.

Dans un premier temps, nous nous intéresserons aux aspects qui ont motivé le développement de ce type de réseau. La réponse à la question « Les Réseaux Intelligents : pourquoi ? » sera constituée de plusieurs volets.

Le premier volet donnera une brève définition de ce qu'on entend par « Réseau Intelligent », afin de fixer les idées et d'assurer la compréhension de la suite de l'exposé. Le deuxième volet reprendra un examen des différents acteurs qui interviennent dans le monde des Réseaux Intelligents, afin de cerner au mieux les rôles qu'ils y jouent et les relations qu'ils entretiennent. Le troisième volet sera consacré à une évocation de l'historique du développement de ces réseaux et ouvrira la perspective offerte à ce développement dans le futur.

Dans un second temps, nous nous intéresserons aux modalités de développement et de mise en œuvre des Réseaux Intelligents. À son tour, la réponse à la question « Les Réseaux Intelligents : comment ? » sera constituée de plusieurs volets.

Le premier volet concernera le cycle de vie d'un service type proposé par un IN. Le deuxième volet exposera l'architecture physique d'un Réseau Intelligent, tandis que le troisième explicitera l'architecture des services IN proprement dits.

1. Les Réseaux Intelligents : pourquoi ?

1.1. Les Réseaux Intelligents : une brève définition

Ce mémoire est, comme nous l'avons dit, réalisé en collaboration avec Alcatel. Pour ne pas nous cantonner dans une seule vision, celle de la définition proposée par Bell-Atlantic que nous avons mentionnée plus haut, de la réalité recouverte par les Réseaux Intelligents, il n'est pas inutile de reprendre ici la définition qu'Alcatel donne du produit que cette firme développe sous ce nom.

Selon Alcatel, donc, il s'agit d'une

> *Architecture réseau ayant pour but de faciliter l'introduction de services nouveaux, complexes, plus rapidement et à un coût moins élevé. Le concept s'applique à tous les types de réseaux et est basé sur une puissance de traitement et un traitement des données étendus.*[3]

Les aspects propres à la création et à la modification de services sont exposés plus loin mais, dès à présent, si l'on reprend les termes des deux définitions, il est intéressant de remarquer que, sous plusieurs aspects, l'accent est mis à la fois sur la qualité et sur la quantité : tant au niveau des services qu'à celui des réseaux et celui des données. Nous reviendrons sur les méthodes utilisées pour correspondre à ces caractéristiques dans la seconde partie de cet exposé consacré aux Réseaux Intelligents, mais il peut être utile, à ce moment-ci, de donner quelques exemples[4] de services offerts par ce type de réseau (les noms de ces services sont parfois plus parlants ou plus explicites en anglais) :

---

[3] *«IN – A Quick Summary»*, in Alcatel Telecommunications Review – *Intelligent Networks : opening the door to new services*, 1st Quarter 1996, p. 4 (traduction personnelle)

[4] Des exemples plus détaillés de services IN sont donnés en annexe.

- Réseau privé virtuel (Virtual Private Network – VPN) ;
- Téléphone gratuit (Freephone) ;
- Services de carte d'appel (Calling Card Services) ;
- Facturation séparée (Alternate Billing) ;
- Vote téléphonique / Appels de masse (Televoting / Mass Calling) ;
- Facturation partagée (Split Charging) ;
- Numéro personnel (Personal Number) ;
- Télécommunication personnelle universelle (Universal Personal Telecommunication) ;
- Appel répété (Repeat Dial) ;
- Blocage d'appel (Blocking) ;
- Sonnerie distinctive (Distinct Ringing) ;
- Etc.

1.2. Les Acteurs[5]

Une fois cette définition posée, l'on peut s'interroger sur les acteurs qui interviennent dans le monde des Réseaux Intelligents. L'examen de ces différents acteurs permettra aussi d'avoir les idées claires lorsque viendra le moment d'exposer les relations qui les animent et les rôles qu'ils jouent les uns par rapport aux autres à la fois dans le développement, la vente, l'exploitation et la gestion des systèmes et des services liés aux Réseaux Intelligents.

1.2.1. Les acteurs classiques

Comme les Réseaux Intelligents sont une des composantes du monde des Télécommunications, l'on peut tout d'abord définir un certain nombre

[5] Cette section puise sa matière dans l'article de Cambré, E., *Intelligent Networks : the key to new services*, in Alcatel Telecommunications Review – *Intelligent Networks : opening the door to new services*, *op. cit.*, p. 14 sqq.

d'acteurs « classiques » de ce domaine d'activités. Les acteurs du monde des Réseaux Intelligents peuvent donc, en premier lieu, être regroupés en niveaux d'une pyramide :

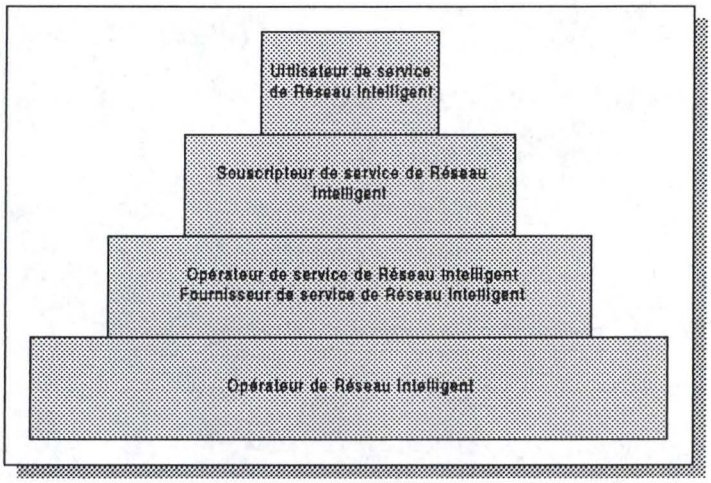


Figure 1 : Acteurs « classiques » d'un *IN*

Chacun des acteurs définis dans les niveaux de la pyramide joue un rôle spécifique :

- L'**Opérateur Réseau** (*Network Operator*) : il s'agit généralement d'une compagnie publique ou privée qui fournit et exploite l'infrastructure dont le Réseau Intelligent a besoin pour supporter la fourniture du service, en collaboration avec l'opérateur PSTN, si elle en est distincte.[6] L'Opérateur Réseau a aussi pour rôle d'être l'interlocuteur contractuel des Opérateurs de Service de Réseau Intelligent (*Intelligent Network Service Operators*), notamment au sujet du trafic des données, de l'utilisation des ressources, *etc.*
- Le **Fournisseur de Service de Réseau Intelligent** (*Intelligent Network Service Provider*) et l'**Opérateur de Service de Réseau Intelligent** (*Intelligent Network Service Operator*) : le Fournisseur de Service de Réseau Intelligent est généralement une personne ou une entité

[6] Nous nous intéressons ici aux services de Réseaux Intelligents accessible *via* le réseau téléphonique, qu'il soit commuté (RTC), RNIS ou mobile.

légale qui offre et fournit le service demandé par l'utilisateur. Cette entité est chargée de la création et du développement du service. L'Opérateur de Service de Réseau Intelligent est chargé de l'exploitation des services. Il est l'interlocuteur contractuel du Fournisseur de Service de Réseau Intelligent, notamment au sujet des caractéristiques du service, des tarifs, *etc.* Il est aussi responsable de l'acquisition et du support aux clients (les *Service Subscribers*), pour lesquels il agit à son tour en tant qu'interlocuteur contractuel, notamment au sujet de la facturation, *etc.*

- Le **Souscripteur de Service de Réseau Intelligent** (*Intelligent Network Service Subscriber*) est une personne ou une société qui souscrit à un service et est enregistré dans la base de données du service. Elle peut prendre une certaine responsabilité dans la définition de certains paramètres du service.
- L'**Utilisateur de Service de Réseau Intelligent** (*Intelligent Network Service User*) : il s'agit de la personne qui effectue les appels téléphoniques qui utilisent un service de Réseau Intelligent.

1.2.2. Les acteurs spécifiques

À ces acteurs traditionnels et bien définis dans le monde des Télécommunications – Opérateurs, Souscripteurs, Utilisateurs –, l'on peut ajouter les Fournisseurs de Systèmes Télécom et d'Applications de Services (*Suppliers of Telecommunication systems and Service applications*), les Fournisseurs de Systèmes de Traitement Électronique des Données (*Suppliers of Electronic Data Processing (EDP) systems*), les Intégrateurs de Systèmes (*System Integrators*), les Consultants et les organismes régulateurs incluant les Institutions de Standardisation. Le rôle et la fonction de ces différents acteurs seront détaillés dans un instant.

Le fait que les acteurs du monde des Réseaux Intelligents ne se réduisent pas aux traditionnels Opérateurs – Souscripteurs – Utilisateurs, et

le fait que leur rôle et les relations qu'ils entretiennent les uns avec les autres évoluent, impliquent de nouvelles exigences quant à la manière dont les Réseaux Intelligents sont développés et implémentés. La gestion du réseau et celle du service changent dans de grandes proportions et se révèlent parfois être un vrai fardeau dans l'utilisation d'un Réseau Intelligent quand il n'a pas été conçu correctement.

1.2.3. Les organismes de standardisation

Un groupe très important, aujourd'hui, dans le monde des Réseaux Intelligents, est constitué par les organismes régulateurs et les autorités de standardisation. Avec la demande sans cesse croissante, de la part du marché, de disposer de davantage d'acteurs indépendants les uns des autres pour participer à la chaîne de développement, il va sans dire qu'émerge un grand besoin d'interfaces strictement définies. Sans elles, il n'y a aucune garantie que les services fonctionneront jamais de manière satisfaisante.

L'ITU,[7] l'ETSI,[8] *etc.*, jouent un rôle majeur dans le processus de standardisation, mais l'industrie IT[9] est elle aussi active, avec des systèmes tels que UNIX, les bases de données SQL, *etc.* La Commission Européenne ou le FCC[10] ont eux aussi beaucoup d'influence, avec certains mécanismes de régulation, qui doivent être reflétés dans la conception et l'architecture des systèmes.

---

[7] *International Telecommunication Union.*
[8] *European Telecommunications Standardization Institute.*
[9] *Information Technology.*
[10] *Federal Communications Commission.*

1.2.4. Les fournisseurs

Du côté des fournisseurs, le monde des Réseaux Intelligents est caractérisé par le fait que, pour le moment, il y a trois groupes actifs majeurs :

- l'industrie des télécommunications ;
- l'industrie du Logiciel ;
- les Consultants.

Alors que les Consultants sont aujourd'hui actifs dans les domaines où les Fournisseurs télécom traditionnels ne l'étaient pas (la plupart des Opérateurs télécom avaient une connaissance interne suffisante pour l'analyse et l'organisation des opérations à effectuer), nous observons maintenant un très net recouvrement des activités liées aux Réseaux Intelligents entre les Fournisseurs télécom et l'industrie du Logiciel, notamment dans le domaine du Traitement Électronique des Données (*Electronic Data Processing – EDP*). Ce chevauchement est souvent source de confusion et chacun de ces acteurs agit au sein du marché selon des règles qui, de façon inhérente, sont différentes.

Cependant, les Réseaux Intelligents ont besoin de composants en provenance de ces deux types d'industries. Si elles sont gérées de façon soigneuse, toutes les parties peuvent y trouver leur bénéfice, et les services proposés pourront devenir un outil de première force pour supporter les activités du marché auquel ils sont destinés : les nouveaux besoins, les nouvelles méthodes et exigences du monde des affaires.

1.3. Contexte du développement des IN[11]

L'historique des Réseaux Intelligents présenté ici sera bref non par manque de place ou de temps, mais parce que l'histoire de ces réseaux est relativement récente : elle date d'une dizaine d'années.

Le concept des Réseaux Intelligents est issu de la compétition. Compétition entre les opérateurs qui les utilisent pour offrir à leurs clients de meilleurs services, dans des délais plus courts, pour attirer de nouveaux clients et gagner le plus de bénéfices possible par l'exploitation de ces services. Cette compétition s'est même étendue aux souscripteurs de services de Réseaux Intelligents qui les utilisent pour améliorer leur position sur le marché en termes de productivité.

Cet aspect de compétition s'est aussi nourri des tendances plus générales que le monde des Télécommunications a vu se développer depuis environ dix ans. Ces tendances peuvent, selon l'analyse d'Alcatel,[12] être catégorisées en trois grands groupes : le marché, la technologie et la demande des utilisateurs.

- Le **Marché** :

  Les tendances au sein du marché peuvent elles-mêmes se subdiviser en trois perspectives d'évolution : libéralisation, privatisation, globalisation.

  - La **libéralisation** dans l'offre de services change le paradigme des rôles précédemment stables des

[11] Ce chapitre s'inspire d'une partie de l'article de Gys L. et Mottram A., *«Intelligent Network : Looking beyond the boundaries of the product»*, in Alcatel Telecommunications Review – *Intelligent Networks : opening the door to new services, op. cit.*, p. 4 sqq.

[12] Gys L. et Mottram A., *«Intelligent Network : Looking beyond the boundaries of the product»*, in Alcatel Telecommunications Review – *Intelligent Networks : opening the door to new services, op. cit.*, pp. 4-13.

Opérateurs et des Fournisseurs de services et de systèmes de télécommunications ;

- La **privatisation** des Opérateurs induit une grande pression sur leurs activités, leur organisation et leur rôle économique ;

- La **globalisation** implique que l'espace d'exploitation n'est plus confiné aux territoires nationaux.

♦ La Technologie :

Bien entendu, le monde des Télécommunications, tout comme celui de l'informatique, est plus que tout autre sujet à l'évolution technologique.

Les Réseaux Intelligents offrent sous cet angle un cas de figure particulier puisqu'ils sont à la croisée des deux mondes : celui des télécommunications et celui de l'informatique. En effet, les *IN* résultent d'une fusion de différentes technologies , qui ont chacune leur propre rythme d'évolution. Les plus importantes parmi celles-ci sont : les réseaux de télécommunications (fixes et mobiles, commutés, ATM, systèmes de transmission, *etc.*) ; les technologies informatiques (caractérisées par un prix du matériel (*hardware*) en constante baisse par nombre de transactions et des performances sans cesse croissantes, caractérisées aussi par un logiciel (*software*) incluant les techniques de programmation orientée objet, les bases de données relationnelles, les interfaces graphiques, les systèmes distribuées, *etc.*).

- La Demande des Utilisateurs :

Les télécommunications ayant pris une part de plus en plus importante dans l'activité des sociétés modernes, il n'est pas étonnant qu'elles jouent un rôle stratégique dans le développement de ces sociétés. À ce titre, leurs utilisateurs auront besoin des meilleurs services pour appuyer cette activité.

Par ailleurs, les utilisateurs sont aujourd'hui beaucoup plus au fait des possibilités offertes par les télécommunications et, par là même, deviennent plus exigeants sur la qualité du service proposé par les Fournisseurs.

1.4. Conclusion : les Réseaux Intelligents répondent à une attente

Les différents aspects des Réseaux Intelligents, tels que nous les avons évoqués ci-dessus, montrent que ceux-ci ont été et sont développés à la fois parce qu'ils sont destinés à générer de nouveaux revenus pour ceux qui les créent, les fournissent, les vendent et les exploitent, et parce qu'ils correspondent à une demande émanant d'acteurs du monde économique et de l'entreprise, qui les considèrent comme un moyen (sinon comme une arme stratégique) de consolider ou d'améliorer leur place sur le marché et, à leur tour, de générer des revenus.

Comme la technologie des Réseaux Intelligents est profitable pour ces deux grands types de partenaires, il y a fort à parier que son développement subira une forte croissance. Les chiffres de cette croissance, depuis une bonne dizaine d'années, permettent de vérifier la validité de ce diagnostic. Les Réseaux Intelligents sont aujourd'hui développés dans la plupart des

réseaux de télécommunications des pays du monde entier et les revenus qui y sont associés sont les suivants :

- Aux U.S.A. : - 20 milliards de dollars en 1996 ;
  - 40 milliards de dollars attendus en 1998 ;

- En Europe : - 6 milliards de dollars en 1996 ;
  - 18 milliards de dollars attendus en 1998.

Par ailleurs, l'on constate aussi que les services liés aux Réseaux Intelligents prennent une part de plus en plus importante de l'offre totale des services de télécommunications. L'on s'attend à ce que cette part passe de 4 % des revenus totaux des services de télécommunications en 1993 à un niveau atteignant 11 % pour l'an 2000. Les avantages du « produit Réseau Intelligent » sur le marché font que le développement des services qui y sont associés ne cesse de croître. Pour conserver ses avantages, ce produit doit correspondre à certaines caractéristiques, qu'il peut être bon d'analyser. Ces caractéristiques forment la conclusion de cette partie de l'exposé, car à la fois elles reprennent les raisons du développement des Réseaux Intelligents et, dans une certaine mesure, elle dictent la façon dont ces Réseaux seront créés et implémentés. Elles constituent donc une bonne transition entre les questions « Pourquoi ? » et « Comment ? » que nous nous posons à l'égard des Réseaux Intelligents. En y regardant de plus près, l'on s'aperçoit que les caractéristiques majeures auxquelles doivent répondre les Réseaux Intelligents sont les suivantes :

- La **rapidité de livraison** n'est plus seulement un souhait, mais bien une exigence du client. Les systèmes et les services doivent être développés dans un délai qui s'exprime en termes de mois, sinon de semaines. Les services doivent être **adaptables** (*customizable*) à la fois par l'Opérateur et par le Souscripteur. Ils doivent aussi pouvoir être *packagés*, par exemple à l'aide de techniques *Plug-&-Play*, notamment par l'équipe de vente lors des négociations.

- La **flexibilité** et la ***scalability***[13] des services nécessitent que leur développement assure un schéma de croissance linéaire pour le logiciel (*software*) et pour le matériel (*hardware*) sous-jacents. La flexibilité implique aussi que l'on soit capable de modifier le service de façon aisée, d'assurer des mécanismes d'adaptation (*customization*) comme parties intégrantes du système et des services.
- La **rentabilité** des services implique que les ressources qui leur sont associées soient bien calibrées, notamment en termes de coût par transaction.
- Les services sont conçus pour assurer le **support d'activités de *business***. Ils ne peuvent donc plus être développés isolément, sans relations avec le Souscripteur ou l'Utilisateur final. Les méthodes de *business* évoluant sans cesse, notamment en fonction des nouvelles possibilités des services de télécommunications, il sera toujours nécessaire de pouvoir effectuer des modifications pour obtenir le comportement spécifique désiré de ces services. Si ce calibrage devait être accompli « à la main », il serait impossible de rendre les services rentables. Il faut donc utiliser des outils CASE.
- Bien entendu, outre toutes ces caractéristiques spécifiques, les services liés aux Réseaux Intelligents doivent continuer à adhérer aux exigences particulières des télécommunications, comme la **disponibilité**, la **facilité d'utilisation**, la **fiabilité**, la **faculté d'être améliorés** (*upgradability*) et de **migrer sans difficulté** – tout cela avec des périodes de support allant jusqu'à quinze ans.

[13] Ce terme n'a pas d'équivalent en français.

2. Les Réseaux Intelligents : comment ?

2.0. Introduction

Après avoir examiné les motivations qui ont présidé au développement des Réseaux Intelligents, nous allons nous intéresser à la manière dont ces réseaux ont été et sont effectivement conçus, réalisés et implémentés.

Pour bien comprendre la façon dont se structure ce concept, il s'agit tout d'abord de déterminer son mode d'utilisation. À cette fin, nous exposerons, en toute généralité, le cycle de vie d'un service *IN*, c'est-à-dire, *grosso modo*, le déroulement chronologique de l'existence de ce service, depuis son développement jusqu'à son éventuel remplacement.

Nous passerons ensuite aux différents éléments qui supportent le service proposé. Il s'agit, en l'occurrence, de l'architecture générale des Réseaux Intelligents. Plusieurs aspects seront à prendre en compte : la façon dont les services sont commutés, contrôlés, gérés, *etc.*

Enfin, nous aborderons l'architecture des services *IN* proprement dits, qui comprend à la fois des aspects de logique en temps réel et des aspects de gestion de service.

2.1. Le cycle de vie du service[14]

Le cycle de vie d'un service offert par un *IN* s'étend de la genèse du service jusqu'à sa complète disponibilité pour l'utilisateur final ou pour son déploiement commercial, et se termine, s'il n'est pas retiré, par son remplacement.

Le développement de la plupart des services peut être réparti dans les quatre phases principales communes à tout projet :

- La phase de conception ;
- La phase de production ;
- La phase d'intégration ;
- La phase de remplacement.

Ces phases constituent un cycle, c'est-à-dire que la dernière phase est suivie par une nouvelle occurrence de la première phase. La nature circulaire de ce cycle est le résultat de l'usage croissant et de l'évolution rapide de Réseaux Intelligents et des services qu'ils offrent, à la fois en termes de quantité et de fonctionnalité.

Le temps de rotation complète du cercle est difficile à définir, dans la mesure où les Réseaux Intelligents ne sont pas depuis suffisamment longtemps sur le marché ; néanmoins, il semble que ce temps de rotation soit aujourd'hui de moins de dix ans. La durée de chaque phase, quant à elle, n'est pas équivalente : par exemple, la phase de développement de service proprement dite dure environ six mois, avec une période d'adaptation (*customization*) de quelques semaines. Ces durées ne peuvent être atteintes

[14] Gys L. et Mottram A., «*Intelligent Network : Looking beyond the boundaries of the product*», in Alcatel Telecommunications Review – *Intelligent Networks : opening the door to new services*, *op. cit.*, pp. 8-11.

qu'au moyen d'architectures bien structurées et de procédures de développement basées sur les outils les plus modernes.

Examinons à présent plus en détail les différentes phases de développement d'un service. Comme nous l'avons dit, celles-ci sont communes à la plupart des projets de développement informatiques. Nous nous attacherons donc plus particulièrement à mettre en exergue les spécificités des projets liés au développement de services associés aux Réseaux Intelligents.

2.1.1. La phase de conception

La phase de conception, dans le développement d'un service *IN* comme dans tout autre projet, est essentielle, dans la mesure où des spécifications peu claires peuvent mener à une grande perte de temps ; or le temps est un aspect clef de ce développement.

Cette phase nécessite donc la collaboration à la fois des experts du développeur du service *IN* et des experts de son client (c'est-à-dire celui qui fournira le service *IN* aux utilisateurs), afin de définir et de tester les nouveaux concepts de services. Dans ce cadre, la demande, en phase de conception, évolue clairement vers une solution prototype de Réseau Intelligent, qui peut être rapidement déployée pour des segments ciblés du marché. Ce prototype fera alors l'objet de la première livraison de la phase de production, que nous examinerons au point suivant.

Toute solution proposée dans le cadre de la phase de conception doit jouir de caractéristiques de rentabilité élevée et de niveaux très étendus d'adaptation (*customization*) pour les utilisateurs individuels. Les Opérateurs pourraient choisir de prototyper dix ou vingt variantes d'un service particulier avant de continuer le processus.

Cette demande mène à l'exigence du concept de Nœud de Service (*Service Node*), où les fonctions *IN* sont rassemblées en une entité unique. Ce Nœud de Service peut n'être pas suffisamment puissant dans le cas d'une implémentation à plus large échelle, mais son efficacité ne peut pas être remise en cause dans le cas du prototypage du service et du test du marché. Dans cette perspective, une voie de migration entre le Nœud de Service et le système de service complet doit être envisagée dès le départ.

2.1.2. La phase de production

La phase de production, au sein du cycle de vie du service, est celle où les fournisseurs ont été traditionnellement les plus actifs. Quelques exemples de services sont décrits en annexe, mais il est, dès à présent, intéressant d'exposer les tendances des « produits *IN* » et des plates-formes qui leur sont associées.

Tout d'abord, il faut mentionner le désir de *contrôle*. De plus en plus de clients exigent davantage de contrôle au sujet de l'évolution de leur *IN* et des services qu'il offre, dans la mesure où ils se rendent compte de l'aspect stratégique, sur le marché, de ce qu'ils considèrent comme une arme très compétitive. Cette exigence de contrôle prend diverses formes, mais elles émanent toutes de la volonté d'être capable de répondre rapidement et adéquatement à une position instable de compétitivité.

Un premier aspect de ce besoin de contrôle se rencontre dans le choix indépendant des éléments *hardware* individuels de la solution *IN*. Ces éléments doivent néanmoins correspondre aux spécifications du monde des Télécommunications, qui sont généralement d'un niveau plus élevé que celles du monde *IT* global. Un autre domaine où le contrôle du client est demandé est celui du développement de service, comme nous l'avons déjà mentionné plus haut, au sujet du prototypage qui est le cœur de la phase de conception.

Un deuxième aspect important est celui du choix de la technologie utilisée, ce qui signifie que l'Opérateur a besoin de tirer profit des meilleures technologies sur le marché pour exécuter le service. Cela inclut les plates-formes *hardware*, mais aussi les systèmes de bases de données et les systèmes de gestion.

En troisième lieu, l'aspect des ressources en termes de capacité à développer les services est à prendre en compte dans la phase de production. Ici, il faut mentionner la méthodologie et les outils propres à *Alcatel*. Les solutions *IN* que cette firme propose sont constituées d'éléments « stabilisés » et d'interfaces qui peuvent être utilisées pour développer et intégrer l'application du service. À ce propos, il peut s'avérer intéressant d'examiner les différentes activités dont l'accomplissement est nécessaire à la création d'un nouveau service. Cette chaîne d'activités est illustrée à la figure suivante :[15]

[15] Les intitulés des composants de la figure sont donnés en anglais pour une plus grande conformité avec la source de l'illustration.

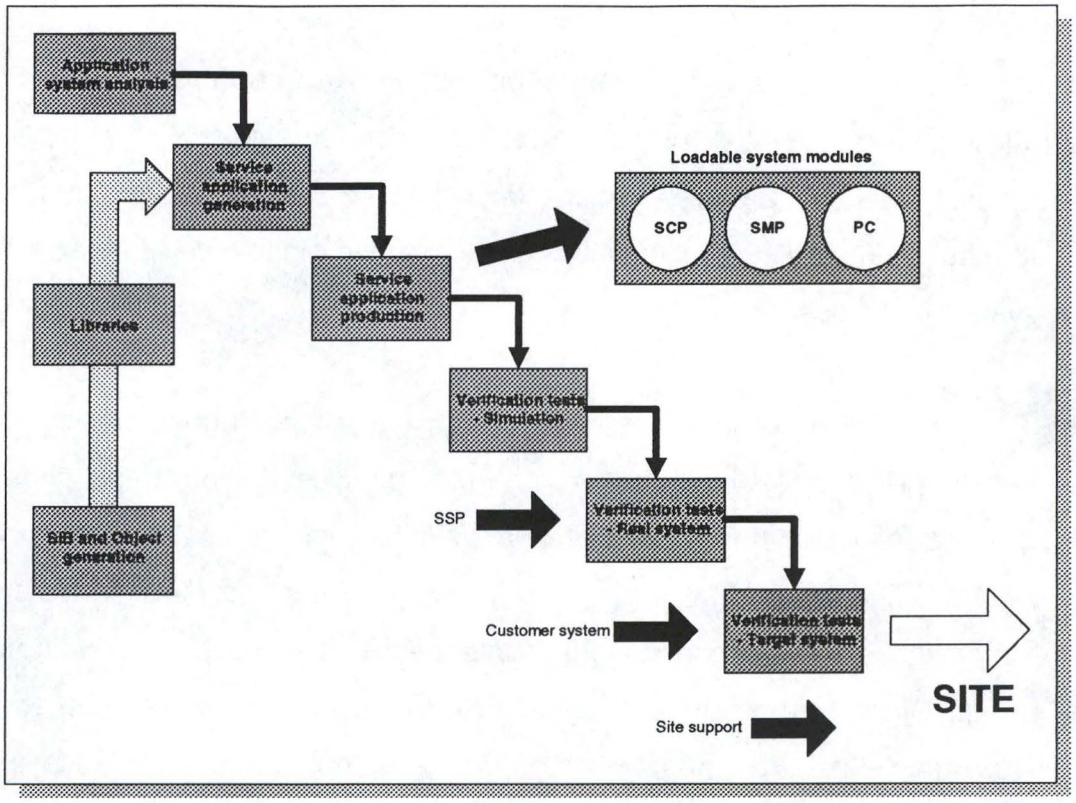


Figure 2 : Chaîne de création d'un service IN

Cet enchaînement d'activités est entièrement supporté par des outils, afin de garantir un cycle de développement le plus rapide et complet possible. Ainsi, en moins de dix ans, le temps de développement est passé de trois ans à moins de six mois aujourd'hui, pour les services les plus complexes, et à quelques semaines pour les services les plus simples.

Les outils sont conçus de telle façon qu'un maximum de blocs de construction existants puissent être réutilisés dans différents services. L'utilisation de ces blocs a elle aussi été maximisée par le développement d'un outil graphique qui offre aux concepteurs une palette des fonctions de base au moyen d'icônes et du langage naturel pour l'entrée des paramètres (*cf. infra*).

2.1.3. La phase d'intégration

La phase d'intégration débute au moment où le « produit service » est prêt à être déployé, au moment où le service doit commencer sa vie « active » dans l'environnement des opérations commerciales.

À cette fin, le Réseau Intelligent doit disposer de composants qui permettront de le gérer, mais aussi d'interfaces, qui permettront une intégration aisée au sein de l'environnement dans lequel il devra fonctionner. L'intégration proprement dite est réalisée pour la plus grande part en coopération avec le client ou avec une entreprise tierce.

2.1.4. La phase de remplacement

Comme la phase de remplacement est souvent le prédécesseur d'une nouvelle phase de conception, il est nécessaire d'y porter son attention en vue d'assurer une voie de migration aussi souple que possible vers le nouveau service amélioré. Cette voie doit aussi être soigneusement étudiée dans la mesure où il faut éviter toute perte de service et d'information relative aux utilisateurs durant la phase de migration.

Avec des services avancés reposant sur des profils d'utilisateurs étendus, construits au cours d'années d'intensives entrées de données, les nouveaux services et les nouveaux systèmes ne peuvent ignorer l'existence de ces données et repartir de zéro. Il est donc nécessaire que le fournisseur s'attache dès le départ à considérer cette phase afin d'éviter toute perte de travail.

2.2. L'architecture physique d'un Réseau Intelligent[16]

L'architecture physique d'un Réseau Intelligent est l'infrastructure qui supporte l'exécution d'un service. Afin de fixer les idées et de rendre la suite de l'exposé plus compréhensible, cette architecture est décrite dans la figure suivante[17] :

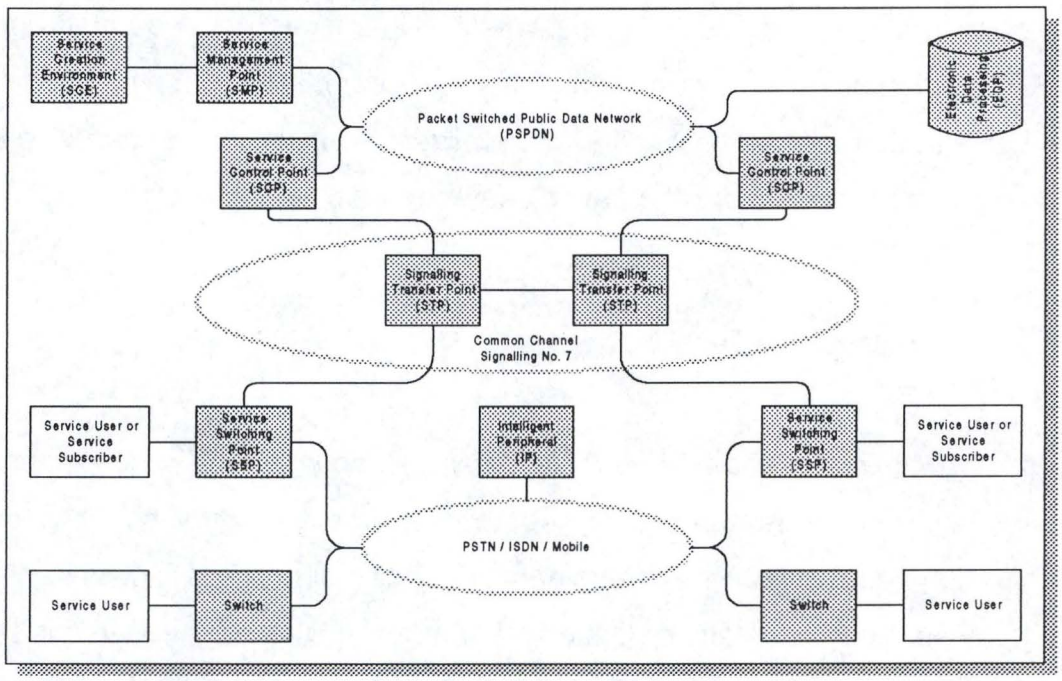


Figure 3 : Architecture physique d'un IN

Cette architecture se compose donc des blocs de construction principaux suivants :

- SSP (Service Switching Point) ;
- SCP (Service Control Point) ;
- SMP (Service Management Point) ;

[16] La référence des informations données ici est Cambré, E., *Intelligent Networks : the key to new services*, in Alcatel Telecommunications Review – *Intelligent Networks : opening the door to new services*, *op. cit.*, pp. 15-17.

[17] Comme pour la figure précédente, les intitulés des composants de la figure sont donnés en anglais pour une plus grande conformité avec la source de l'illustration.

- IP (Intelligent Peripheral) ;
- SCE (Service Creation Environment).

À chacun de ces blocs correspondent des fonctions spécifiques que nous allons examiner.

2.2.1. SSP : Service Switching Point

Le SSP est le point d'accès aux services *IN*. Un utilisateur du service, à partir de n'importe quel endroit sur le réseau, peut faire usage du service en en formant le préfixe ; le réseau peut atteindre le service au moyen d'un routage d'appel classique.

Une fois qu'un appel de service *IN* est reçu, le SSP l'identifie (par exemple, par l'analyse des chiffres formés (*Digit Analysis*)) comme un appel *IN*. Ensuite, le rôle du SSP consiste à découvrir quel service est demandé, où le service est physiquement localisé (c'est-à-dire : dans quel SCP) et quelle information supplémentaire doit être agrégée au sujet de l'appel (par exemple, l'identification de la ligne de l'appelant (*Calling Line Identification*)).

Le SSP invoquera alors le service pour cet appel particulier dans le SCP (*Service Control Point*). À partir de ce moment, la logique de service dans le SCP prend le contrôle de l'appel.

2.2.2. SCP : Service Control Point

Le rôle du SCP est principalement d'assumer les fonctions temps-réel et l'intelligence d'un service. Il contient aussi la base de données associée au service, qui peut elle-même être localisée dans un nœud à distance du SCP.

Quand la logique de service dans le SCP est activée, elle traite les diverses données reçues avec le message d'activation du service. Par exemple, une simple action serait de traduire le numéro appelé en un autre numéro E.164 (*Public Network Numbering Standard*). Selon le scénario du service, la logique de service commandera au SSP et à l'IP (*Intelligent Peripheral*) l'exécution de certaines actions[18] :

- SSP :
  - Établir une connexion vers une destination E.164 ;
  - Mettre à jour le contenu de l'enregistrement de tarification ;
  - Être à l'écoute, contrôler (*monitor*) les événements.

- IP :
  - Envoyer de l'information (messages audibles, texte lisible affiché sur le poste de l'utilisateur) ;
  - Recevoir de l'information (signaux DTMF) en provenance de l'utilisateur du service.

Le SSP et l'IP peuvent eux-mêmes renvoyer des notifications à la logique de service, telles que :

- En provenance du SSP : les événements reçus (réponse, ...) ;
- En provenance de l'IP : l'information reçue de l'utilisateur du service.

2.2.3. IP : Intelligent Peripheral

L'IP représente un élément important de l'architecture d'un Réseau Intelligent. Il peut être considéré comme un mécanisme jouant un rôle d'intermédiaire entre la logique de service et l'utilisateur du service.

[18] *Cf.* les exemples en annexe.

Beaucoup de services demandent que, pendant l'appel relatif au service, l'utilisateur soit guidé afin qu'il exécute certaines actions et/ou obtienne de l'information qui doit lui être fournie. L'IP envoie des messages audibles ou du texte à l'utilisateur du service et transmet les informations fournies par celui-ci au SCP.

2.2.4. SMP : Service Management Point

Le SMP centralise toutes les fonctions nécessaires à la gestion d'un Réseau Intelligent et des services *IN* qui lui sont associés :

- Gestion de la performance ;
- Gestion des alarmes ;
- Gestion de la configuration ;
- Gestion des accès ;
- Mesures et génération de statistiques.

Comme l'on peut s'y attendre, les services *IN*, puisqu'ils sont centralisés, nécessitent la gestion de grandes quantités de données. Ces données peuvent être réparties sur divers nœuds physiques, *i.e.* le SMP lui-même et les SCP qui lui sont reliés. Ces données doivent bien entendu être correctes et consistantes à l'échelle du réseau.

Le SMP est également l'interface directe des terminaux de gestion à partir desquels des commandes peuvent être données (par exemple, *créer un souscripteur*) et auxquels des rapports peuvent être envoyés (par exemple, des *statistiques* ou des *alarmes*).

Les opérations de gestion peuvent être exécutées par le fournisseur de service aussi bien que par le souscripteur de service : le concept de Réseau Intelligent autorise les profils de service individuels, donc le

souscripteur de service doit pouvoir être à même d'adapter, dans les limites de sa souscription contractuelle, son propre profil.

2.2.5. SCE : Service Creation Environment (1)

Comme nous l'avons dit, la capacité de développer et de déployer des services *IN* rapidement est cruciale. Le SCE est la plate-forme de développement des services *IN* permettant de garantir cette rapidité. Il contient tous les outils et les interfaces homme-machine pour développer et modifier aisément les services.

Le SCE a trois fonctions principales :

- Le développement des SIB (Service Independent Building Blocks) ;
- Le développement des services ;
- L'adaptation (customization) des services.

Nous reviendrons au SCE après avoir examiné l'architecture propre aux services *IN*.

2.3. L'architecture des services IN

Chacun des services offerts se compose de deux blocs fonctionnels principaux :

- La logique de service en temps réel ;
- Le bloc de gestion du service.

Nous nous intéresserons successivement à ces deux blocs, avant de passer aux SIB (*Service Independent Building Blocks*).

2.3.1. La logique de service en temps-réel

Localisée physiquement au sein du SCP, la logique de service communique en temps réel avec le SSP et l'IP par le biais du protocole INAP. Elle reçoit de l'information en provenance du SSP au sujet des événements générés par l'appel *IN* et en provenance de l'IP au sujet des entrées fournies par l'utilisateur du service.

La logique de service a aussi accès à la base de données du service, qui peut être située au sein du SCP lui-même ou ailleurs.

Pour chaque appel, un enregistrement est créé par la logique de service, dans lequel toutes les données pertinentes relatives à l'appel sont inscrites. L'enregistrement est transmis au SCP à la fin de l'appel.

La logique de service envoie également des commandes au SCP et à l'IP afin d'établir et de couper les connexions, d'envoyer de l'information (messages vocaux, textes) à l'utilisateur du service, de recevoir de l'information en provenance de cet utilisateur, de surveiller les événements générés par le réseau, d'envoyer des informations de tarification, *etc.*

2.3.2. Le bloc de gestion du service

Le bloc fonctionnel de gestion du service exécute les commandes de gestion données à partir des terminaux graphiques de gestion du fournisseur ou du souscripteur du service. Il est physiquement localisé au sein du SMP.

Le fournisseur de service peut accéder au système pour toutes les activités de gestion par lesquelles il est concerné :

- Pour être informé de toute alarme de service, de toute anomalie ;
- Pour générer des statistiques sur l'utilisation du service et ses caractéristiques ;
- Pour activer une surveillance sur des souscripteurs de service individuels ;
- Pour créer et effacer des souscripteurs de service avec des profils adaptés ;
- Pour exécuter des fonctions relatives à la comptabilité ;
- Pour exécuter des fonctions relatives à la sécurité.

Les caractéristiques qui peuvent être assignées individuellement aux souscripteurs incluent les éléments suivants :

- Routage de l'appel dépendant de l'origine ;
- Routage de l'appel dépendant de l'heure ;
- Message d'accueil ;
- Commission sur l'appel entrant ;
- Restriction du nombre d'appels entrants.

2.3.3. SIB : Service-Independent Building Blocks

Comme on l'aura sans doute remarqué, beaucoup de caractéristiques de service sont communes à divers services. Cette constatation a mené à l'idée de créer une bibliothèque d'éléments communs, les SIB (*Service Independent Building Blocks*), qui peuvent être aisément assemblés afin de fournir n'importe quelle combinaison de caractéristiques désirée. Ce concept est appliqué à la fois à la logique de service en temps réel et au bloc de gestion du service.

Voici quelques exemples de ces caractéristiques communes à plusieurs services :

- Routage de l'appel dépendant de l'heure, de la date ou du lieu d'origine de l'appel ;
- Plan de numérotation privé ;
- Validation d'un PIN (*Personal Identification Number*).

2.3.4. SCE : Service Creation Environment (2)

Comme nous l'avons mentionné plus haut, le SCE a trois fonctions principales :

- Le développement des SIB (Service Independent Building Blocks) ;
- Le développement des services ;
- L'adaptation (customization) des services.

Passons en revue ces trois fonctions.

2.3.4.1. Développement de SIB

Il arrive qu'un client demande un service dont les caractéristiques sont telles qu'elles ne sont pas couvertes par la bibliothèque de SIB existants. Dans ce cas, un nouveau SIB doit être développé. Ce développement est basé sur une analyse profonde des exigences du client. Le *designer* définit le modèle logique de données du SIB ; les outils graphiques assistent le codage, qui est effectué au moyen d'un SDL (*System Description Language*). Le nouveau SIB, une fois testé, validé et stabilisé, est inclus dans la bibliothèque.

2.3.4.2. Développement de services

Quand un service doit être créé, une sélection des SIB appropriés est effectuée, en fonction des exigences du client. Des outils dédicacés sont utilisés pour assembler les SIB afin de générer :

- La logique de service en temps réel et la structure de la base de données associées au service ;
- Le modèle physique de données destiné à être exécuté au sein du SCP ;
- Le bloc de gestion du service, contenant les commandes de gestion et la structure de base de données relationnelle destinées à être exécutées au sein du SMP et des terminaux de gestion.

2.3.4.3. Adaptation (*customization*) de services

L'utilisation de l'outil décrit plus haut permet la composition de nouveaux services et l'adaptation de services existants. Les SIB peuvent être sélectionnés à partir de la bibliothèque du client afin de configurer de nouvelles combinaisons de caractéristiques de service. Il y a bien entendu des restrictions au niveau de la combinaison des SIB utilisés dans un service, aussi bien qu'au niveau de l'ordre dans lesquels ils sont utilisés. L'outil d'adaptation du service informe automatiquement l'utilisateur de toute incompatibilité.

3. Conclusion de la première partie

Cette première partie, comme attendu, nous a permis de répondre aux questions « pourquoi » et « comment » que nous nous posions au sujet des Réseaux Intelligents.

Au niveau du « pourquoi », après avoir vu que la notion d'intelligence au sein d'un réseau a été introduite dans la perspective d'accroître les capacités de services de ce réseau, c'est-à-dire d'augmenter le nombre et la complexité des services, d'en réduire les temps de développement et les coûts, nous nous sommes intéressé aux différents acteurs qui interviennent dans cette partie du monde des télécommunications. Il s'est agi ensuite d'évoquer le contexte de l'évolution des Réseaux Intelligents.

Au niveau du « comment », nous avons montré que le cycle de vie d'un service IN se découpait en les quatre phases de conception, production, intégration et remplacement. L'architecture physique d'un Réseau Intelligent se compose, elle, de SSP (*Service Switching Point*), SCP (*Service Control Point*), IP (*Intelligent Peripheral*), SMP (*Service Management Point*) et SCE (*Service Creation Environment*), éléments dont nous avons vu le rôle joué par chacun au sein du réseau. Nous avons ensuite mentionné la découpe de l'architecture des services IN en logique de service en temps-réel et bloc de gestion de service, avec un aperçu des fonctionnalités et du développement des SIB (*Service-Independent Building Block*), en relation avec le SCE (*Service Creation Environment*).

Partie II. Internet & les Réseaux Intelligents

0. Introduction

De manière analogue à celle qui caractérisait l'exposé sur les Réseaux Intelligents, objet de la première partie de ce mémoire, la présentation relative à Internet se fera selon une double perspective.

Tout d'abord, après une brève définition, nous rappellerons rapidement les motivations historiques qui ont présidé au développement du Réseau des réseaux. Nous analyserons ensuite certaines des tendances actuelles dont il fait l'objet et les raisons qui nous poussent à nous y intéresser dans le cadre de ce mémoire.

Nous passerons alors à un exposé plus technique de certaines des caractéristiques d'Internet, avec un aperçu des protocoles principalement utilisés (IP, TCP et HTTP) et des langages de description (HTML) et de programmation (Java) souvent associés à Internet.

Ces différents aspects seront abordés dans l'objectif d'exposer les techniques et les outils qui seront utilisés dans le cadre de la troisième partie de cet ouvrage. Dans cette perspective, il est évident que nous effectuons un choix parmi un ensemble de points susceptibles d'être développés. C'est ainsi que, au sujet d'Internet, nous nous intéresserons principalement au *World Wide Web* (*WWW*) et aux techniques qui y sont associées, bien qu'Internet, dans sa globalité, ne s'y réduise en aucune manière. Nous renvoyons à l'immense documentation relative au Réseau des réseaux pour les nombreux points qui ne seront pas abordés dans l'espace de ce mémoire.

1. Internet : pourquoi ?

1.1. Internet : une brève définition

D'un point de vue technique et fonctionnel, *Internet est avant tout un moyen de communications et d'échanges. Le support physique de cette communication est* [...] *l'ordinateur. Il est présent à tous les niveaux, que ce soit à l'émission, à la réception et à tous les stades intermédiaires de la transmission. Mais pas particulièrement pour ses fonctions avancées ou sa puissance de calcul puisque l'on pourrait utiliser virtuellement tous les types d'ordinateurs passés et à venir. Ce que l'on utilise dans l'ordinateur, ce sont ses capacités d'entrée/sortie (clavier, écran), offrant plus ou moins de confort en fonction de la configuration installée (disque dur, imprimante, logiciels), et de communication. Et, après avoir défini un mode de communication entre ordinateurs et les moyens à mettre en œuvre pour que chaque type d'ordinateur ait la possibilité de les utiliser, on a simplement interconnecté entre eux ces ordinateurs de façon à créer un gigantesque réseau dans lequel, à la manière du téléphone, chaque point peut communiquer avec un autre point.*[19]

Comme nous l'avons dit, nous nous intéresserons principalement au *WWW*. Afin de donner une idée claire et distincte de la réalité que nous évoquons, nous reprenons la définition de la *toile* telle qu'elle est donnée dans le cours de *matières approfondies de Téléinformatique* de M. van Bastelaer :

> *Le **WWW** ou World-Wide Web (Toile d'araignée mondiale), plus couramment appelé le Web, est d'abord un concept inventé par le CERN (Laboratoire Européen de Physique Nucléaire) en 1992 et qui permet à une entité cliente d'avoir accès à des documents hypermedia résidant sur un ou plusieurs serveurs*

[19] Lips, B., *Internet en Belgique*, Best Of Éditions, Bruxelles, 1996, p. 24.

> *communiquant au moyen des protocoles Internet et de présenter ces documents à des utilisateurs humains ; c'est aussi, et peut-être principalement, devenu le nom donné à l'ensemble des services basés sur ce concept et disponibles sur le réseau Internet mondial.* [...][20]

Nous retiendrons comme élément principal de cette définition l'aspect Client/Serveur des transactions effectuées sur le *Web*, que nous reprendrons dans la troisième partie de ce mémoire.

1.2. Internet : un bref historique

Beaucoup d'ouvrages et d'articles s'étendent sur la genèse du Réseau des réseaux. Nous n'y revenons ici que pour mémoire et par souci de complétude. A cette fin, nous nous inspirons de l'historique succinct présenté par B. Lips dans son ouvrage *Internet en Belgique*.

1.2.1. Origine militaire

La naissance d'Internet est à replacer dans le contexte du monde de l'informatique tel qu'il se présentait voici un bon quart de siècle, aux États-Unis :

> *À cette époque, l'environnement informatique est constitué de gros systèmes centralisés, puissants mais volumineux et coûteux. Chaque site sert plusieurs centaines d'utilisateurs, qui utilisent des terminaux pouvant être géographiquement distants de l'ordinateur central. La connexion entre les terminaux et l'ordinateur se fait via des connexions locales ou même distantes via des lignes téléphoniques. Travaillant pour la plupart en mode texte, le volume d'information à transmettre entre le terminal et l'ordinateur est faible, ce qui autorise l'usage de modems « lents ». Comme les ressources informatiques sont partagées, on travaille le plus*

[20] van Bastelaer, Ph., *Chapitre WWW - Niveau 7 : Le World-Wide Web, HTTP et HTML*, avril 97, p. 3, *in* Nachtergaele, V. et van Bastelaer, Ph., *Cours de compléments de téléinformatique*, Deuxièmes Licence et Maîtrise en Informatique, FUNDP, septembre 1997.

> *souvent en différé (en batch), envoyant des instructions qui seront exécutées avec un certain délai.* [...]
>
> *Dans ce contexte de partage de ressources et de délocalisation, on en est vite venu à penser que si on pouvait connecter des terminaux à distance, on pourrait également travailler sur plusieurs ordinateurs distants. Pour cela, il suffisait de connecter les sites entre eux (connexion physique) et de régler le problème de la circulation des informations (connexion logique).*
>
> *Sous l'égide du département de la Défense U.S., un projet d'étude est alors mis en place.* [...] *Ce projet doit permettre de réaliser l'interconnexion de plusieurs ordinateurs (de marque, type et système d'exploitation différents) et la communication et l'échange d'informations à partir de n'importe quel point de ce réseau. L'idéal étant aussi d'assurer l'indépendance complète vis-à-vis des matériels (ordinateurs, logiciels), du type et de l'état des connexions qui les relient.*
>
> *En 1969, les travaux aboutissent à l'élaboration d'une série de protocoles, connus sous le nom générique de TCP/IP, définissant les modes d'identification des ordinateurs sur un réseau et les modalités de communication entre eux.*[21]

La mise en œuvre de cette suite de protocoles donna naissance au réseau ArpaNet. On le voit, l'origine du Réseau des réseaux est militaire. L'objectif était d'assurer la continuité des moyens de communication, même en cas d'agression nucléaire et de destruction d'éléments du réseau.

> *Aussitôt mis en œuvre sur une trentaine de sites U.S., le réseau ArpaNet devient bien vite le prototype de ce qui sera plus tard Internet. Rapidement séparé de sa composante militaire, jugée trop « sensible » par l'armée et qui formera MilNet, le succès d'ArpaNet est relativement immédiat et l'on assiste très vite à un accroissement des ordinateurs interconnectés. Malgré des tentatives provenant d'autres constructeurs (comme IBM ou DEC) ou d'autres organismes (comme l'ISO) d'établir d'autres standards d'interconnexion à grande échelle, les protocoles TCP/IP se sont définitivement installés et sont toujours utilisés aujourd'hui.*[22]

---

[21] Lips, B., *Internet en Belgique, op. cit.*, pp. 28-29.
[22] Lips, B., *Internet en Belgique, op. cit.*, pp. 29-30.

1.2.2. Relève universitaire

Une fois les bases du réseau jetées et son aspect militaire évacué, les universités en ont poursuivi le développement :

> *Dans le monde universitaire, l'adhésion a été immédiate. Au sein des « centres de calculs » [...], on s'est attelé à installer les connexions entre sites. Intégrés à Unix, le système d'exploitation disponible gratuitement et largement déployé au sein de la communauté universitaire, les protocoles TCP/IP ont servi de base à de nombreux cours, travaux ou exercices de programmation. C'est ainsi que de nombreux étudiants ont développé la plupart des outils qui forgent aujourd'hui les services d'Internet.* [...][23]

1.2.3. Poursuite du développement

D'autres organismes publics et privés ont ensuite assuré l'avenir du réseau :

> *Une nouvelle impulsion fut donnée par la Fondation pour les Sciences U.S. (NSF) qui, lorsqu'elle décida d'interconnecter 5 de ses super-calculateurs, choisit TCP/IP. Adoptant d'emblée un débit de connexion en rapport avec la puissance des super-calculateurs et en autorisant les raccordements des autres réseaux scientifiques pour leur donner accès à ses machines, la NSF a créé le modèle fédéral et fédérateur d'Internet : un ensemble de réseaux régionaux, regroupant des réseaux de campus, interconnectés via une puissante dorsale. Le succès du NSFNet fut tel que de nombreux utilisateurs vinrent encore s'ajouter, délaissant petit à petit ArpaNet. De son côté, le NSFNet devint une des composantes principales d'Internet, véhiculant la grosse majorité du trafic entre les différents sites, abandonnant presque son rôle initial vis-à-vis des super-calculateurs au profit de la multitude de sites générés par l'explosion et la montée en puissance de l'informatique « personnelle ».*

---

[23] Lips, B., *Internet en Belgique, op. cit.*, p. 30.

> *En 1994, la NSF souhaitant se désengager financièrement pour subsidier d'autres projets de recherche et ayant estimé – à juste titre – que le prototype était suffisamment mûr, engagea des tractations avec différents acteurs publics et privés pour la mise en place d'une infrastructure de remplacement. C'est aujourd'hui chose faite puisque la dorsale est gérée conjointement par MCI, Sprintlink et ANS.*[24]

C'est aussi dans le courant du début des années 90 que le nombre des utilisateurs d'Internet a commencé à croître de façon exponentielle, notamment à cause de l'émergence du réseau en Europe et du développement du *World-Wide Web*.

1.3. Des tendances actuelles d'Internet

Nous ne prétendrons pas ici faire un exposé exhaustif – qui le pourrait ? - des perspectives dans lesquelles semble s'orienter le développement du Réseau des réseaux mais, outre l'accroissement exponentiel du nombre de ses utilisateurs que nous avons déjà mentionné plus haut, nous désirons faire part au lecteur de deux tendances qui semblent s'inscrire parfaitement dans les lignes directrices de ce mémoire. La première est le commerce électronique, la seconde est la technologie *Push*. Pour l'exposé de ces tendances, pour varier nos sources d'information, et puisque Internet n'est plus seulement réservé à une certaine couche, étroite, de la société, nous nous sommes servi d'articles parus dans des revues destinées au grand public ces derniers mois, ou accessibles *on-line*.

1.3.1. Le commerce électronique

Le commerce électronique, au sens où nous l'entendons ici, représente la possibilité de choisir, passer commande et régler le prix d'un bien *via* Internet. L'éventail des marchandises proposées va des ordinateurs

[24] Lips, B., *Internet en Belgique*, *op. cit.*, pp. 30-31.

et de leurs composants aux fromages, et des vêtements aux fleurs, en passant par les meubles et les bijoux :

> *Les catégories de biens concernées sont très nombreuses et ne cessent de se diversifier. Déjà les grands groupes de vente par correspondance sont présents sur le réseau (La Redoute et 3 Suisses, par exemple). Ils ont été rejoints par quelques grands de la distribution (Fnac), et les cybermarchés prolifèrent de jour en jour.*[25]

Si ce marché tarde encore un peu à décoller en Europe (*25 % des sites* [européens] *ont été rentables en 1997, le reste s'attendant à une rentabilité à moyen terme* ; [aux Etats-Unis] *58 % des sites actifs depuis plus de trois ans ont atteint la rentabilité*)[26], les analystes s'accordent cependant pour affirmer que ces *nouvelles techniques de vente*[27] généreront des revenus non négligeables :

> *[...] les ventes réalisées grâce au commerce électronique en France devraient passer de 688 millions de FRF en 1997 à 57 milliards en 2001.*[28]

Au niveau de la mise en œuvre d'un système de commerce électronique, plusieurs aspects sont à prendre en compte :

- La création du site Internet : celui-ci présente les produits disponibles à la vente et doit rencontrer adéquatement les désirs des clients afin de générer l'envie puis la décision d'achat ;
- L'hébergement du site Internet : le catalogue des produits doit être aisément accessible (par exemple *via* une adresse représentative et/ou facile à retenir), – et il est nécessaire qu'il soit localisé sur un serveur pouvant gérer un nombre suffisant d'accès simultanés sans

---

[25] *L'an 1 du commerce électronique*, *in* L'Ordinateur individuel, n° 92, février 1998, p. 30.
[26] Lévy, M., *Pionniers - Ils ont adopté le commerce électronique*, *in* PC Professionnel, n° 8, juin 1998, p. 144.
[27] Lévy, M., *Pionniers - Ils ont adopté le commerce électronique*, *op. cit.*, p. 143.
[28] Billaut, J.-M., cité par Lévy, M., *in Pionniers - Ils ont adopté le commerce électronique*, *op. cit.*, p. 143.

provoquer de ralentissements susceptibles d'entraver le *shopping* du client potentiel ;

- La transaction commerciale et financière de la commande : celle-ci comprend *l'authentification du marchand, l'affichage du formulaire de commande et le calcul du prix final, le paiement sécurisé, l'envoi d'un reçu électronique et le suivi des commandes.*[29]

Le développement d'un tel type de commerce aura aussi des répercussions au niveau socio-économique :

> *L'essor des échanges électroniques devrait notamment entraîner l'émergence de nouveaux modèles économiques et de nouvelles règles du jeu de la concurrence.*[30]

1.3.2. La technologie Push

Le *Push* est une technologie apparue, sous sa forme actuelle, relativement récemment sur Internet. Son nom fait opposition au mode habituel du Réseau, qui consiste, pour l'utilisateur, à aller chercher (en anglais *to pull*, – tirer) l'information là où elle se trouve. Il s'agit maintenant que cette information soit *poussée* (*push*) vers la personne qui en a besoin :

> *Le Push, connu aussi sous le nom de Webcasting, permet aux utilisateurs de recevoir passivement de l'information diffusée, plutôt que de sonder activement le Web ou les sources intranet à la recherche de l'information. La technologie Push permet aux récepteurs de spécifier le type d'informations qu'ils désirent recevoir à partir d'un menu de sources. Ces informations sont ensuite affichées sur l'écran de son ordinateur.*[31]

---

[29] Bedin, F., *Vendez sur le Web : l'opérateur télécom s'occupe de tout !, in* PC Professionnel, n° 9, juillet/août 1998, p. 29.
[30] *L'an 1 du commerce électronique, in* L'Ordinateur individuel, *op. cit.*, p. 30.
[31] Radosevich, L. *Pushing it, in* CIO Magazine, 1er mai 1997 (magazine archivé sur le Web à l'adresse http://www.cio.com/ciomag/archive/050197_push.html), traduction et adaptation personnelles.

Les experts estiment que ce nouveau type d'interaction constitue pour Internet une révolution d'une ampleur analogue ou supérieure à celle de l'introduction du concept *Web* (c'est-à-dire de l'interface que constitue le *Web*). Cette révolution semble devoir être la source de revenus importants :

> *Le marché de la technologie Push générera, en l'an 2000, un revenu de 5,7 milliards $ sur un total de 19,1 milliards $ pour le marché du Web en général.*[32]

Le développement de la technologie *Push* affecte le monde des réseaux à plusieurs niveaux, celui d'Internet, mais aussi celui des Intranets d'entreprise :

> *Les services de news d'Internet ne sont qu'une forme du Webcasting. Les entreprises peuvent utiliser les technologies Push sur leurs intranets pour distribuer à propos aux employés des informations relatives à la société.*[33]

Au niveau des particuliers et d'Internet, le *Push* est surtout lié aux loisirs et à l'information générale, tandis qu'au niveau des entreprises et de leurs intranets, le *Push* est davantage attaché à une utilisation professionnelle, nécessitant des informations plus ciblées (par exemple, les cours de la bourse). Dans les deux cas, le contenu de ce qui est envoyé peut être choisi et déterminé au mieux des souhaits de l'utilisateur, et son affichage peut lui aussi être configuré :

> *Au lieu de livrer la totalité des rubriques, votre journal virtuel ne vous envoie que les canaux d'information auxquels vous vous êtes inscrit. Puisque chacun de ces canaux peut être personnalisé pour rencontrer vos besoins informationnels, vous n'avez pas à craindre d'être inondé d'informations sans intérêt pour vous.*
>
> *Quand un nouveau contenu arrive à votre ordinateur, l'information peut être affichée d'une multitude de façons : dans*

---

[32] Radosevich, L. *Pushing it, in* CIO Magazine, *op. cit.*
[33] Radosevich, L. *Pushing it, in* CIO Magazine, *op. cit.*

> *une petite fenêtre au bas de votre écran, comme un économiseur d'écran actif lorsque votre PC est inutilisé, etc.*[34]

L'idée d'un contenu « poussé » vers son utilisateur n'est pas si nouvelle, mais on assiste aujourd'hui à un phénomène de convergence et de standardisation des différents procédés qui reposent sur la philosophie du *Push* :

> *L'idée de base sous-jacente au modèle de livraison Push ne date pas d'hier. Bien que les diffusions d'e-mails en soient sans doute les exemples les plus familiers, des logiciels et des services propriétaires ont aussi été utilisés pour distribuer automatiquement un certain nombre de choses, des news aux mises à jour de logiciels. Mais le phénomène le plus important concerne la convergence entre la technologie Push et les ressources immenses d'Internet, couplées à sa grande pénétration.*[35]

Par ailleurs, le *Push* fonctionne selon le modèle général des transactions sur le *Web*, le Client/Serveur :

> *Le logiciel qui reçoit et affiche l'information sur le PC s'appelle un client Push. De même que d'autres logiciels clients, comme les browsers Web, les clients Push ne représentent qu'un des côtés d'un processus Client/Serveur. De l'autre côté se trouvent les serveurs push, qui sont responsables de l'acheminement des canaux d'information vers les clients.*[36]

Aujourd'hui, *via* la technologie *Push*, il est même possible de ne plus fournir seulement des informations, mais aussi du contenu exécutable :

> *Outre la distribution de contenu, des formes émergentes de technologie Push permettent de distribuer des applications. Certains logiciels de Push clients incluent un « tuner » qui récupère l'information au départ de canaux sur un serveur.*

---

[34] Stanek, W. R., *Pushing the Envelope with Push Technology, in* PC Magazine, 23 septembre 1997 (magazine archivé sur le Web à l'adresse http://search.zdnet.com/pcmag/issues/1616/pcmg0032.htm), traduction et adaptation personnelles.

[35] Radosevich, L. *Pushing it, in* CIO Magazine, *op. cit.* Au sujet de la standardisation , le lecteur intéressé peut notamment consulter le site http://www.microsoft.com/standards/cdf-f.htm qui présente la solution proposée par Microsoft pour résoudre le problème des différentes technologies *Push* propriétaires, basée sur le CDF (*Channel Definition Format*).

[36] Radosevich, L. *Pushing it, in* CIO Magazine, *op. cit.*

> *Cette information peut contenir des applets Java qui fournissent à l'utilisateur des fonctionnalités de programme sans la nécessité pour lui d'installer quoi que ce soit.*
>
> *En « poussant » des applets Java en même temps que du contenu, les entreprises peuvent plus aisément déployer des applications interactives pour ordinateurs, pagers, kiosques d'informations ou quelque plate-forme que ce soit.*[37]

Bien que le développement de la technologie *Push* soit prometteur, il ne va pas sans poser un certain nombre de problèmes, dont le moindre n'est pas que le *Push* est en réalité ... une fausse révolution ![38] D'autres sont soulevés par différents analystes :

> *Comme toute nouvelle technologie merveilleuse, le Push présente plusieurs risques. Des mises à jour de contenu à partir d'Internet, non correctement gérées, peuvent bloquer les firewalls des entreprises ; des problèmes de sécurité peuvent apparaître si le contenu distribué recèle du code exécutable.*[39]

1.4. Conclusion : pourquoi s'intéresser à Internet dans le cadre des IN ?

Du fait de sa grande pénétration, des facilités qu'il offre en termes d'accès, de disponibilité, de coût, *etc.*, Internet se présente comme un moyen valable d'interaction avec un système distant, – c'est en partie cela, nous l'avons vu, qui a motivé son développement. Or il se fait qu'avec les grandes capacités d'adaptation (*customization*) des Réseaux Intelligents laissées à leurs clients par les fournisseurs, un tel moyen d'accès distant n'est pas à négliger.

---

[37] Radosevich, L. *Pushing it, in* CIO Magazine, *op. cit.*

[38] *Ce qui est considéré comme du Push est en fait du pull, l'ordinateur* [client] *recherchant lui-même la nouvelle information.* Plus de détails sont donnés dans la 3e partie de ce mémoire.

[39] Radosevich, L. *Pushing it, in* CIO Magazine, *op. cit.*

C'est, notamment, ce qui motive ce mémoire. On a en effet pensé qu'une partie des activités de surveillance d'un Réseau Intelligent (ce qu'on appelle *monitoring*) pourrait se faire via Internet.

La troisième partie du travail qu'on a sous les yeux n'est rien d'autre qu'une tentative d'implémentation d'un prototype d'une application qui remplirait cet objectif.

2. Internet : comment ?

2.0. Introduction

Nous donnons ci-après un aperçu des caractéristiques d'Internet, et de certains protocoles et langages qui lui sont fréquemment associés. Partant du rappel de quelques principes généraux d'interconnexion, en particulier au niveau de la couche IP, nous exposons les protocoles IP, TCP et HTTP, ainsi que les langages HTML et Java.

Nous nous devons de mentionner ici que nous avons écourté la présentation des protocoles de certaines de ses parties les plus techniques. Il s'agit en général des primitives de service et du format des unités de données de chaque protocole examiné. Le lecteur intéressé (et patient) trouvera ces parties en annexe ; – nous avons toutefois indiqué par un titre dans le corps du texte l'endroit où ces présentations ont lieu de se trouver, afin d'assurer la cohérence de l'exposé.

2.1. Caractéristiques d'Internet

Ainsi qu'on l'a dit et répété, Internet est le Réseau des réseaux. Cela signifie qu'Internet est un modèle d'interconnexion de sous-réseaux. Comme ce mémoire se place dans la perspective des télécommunications, il n'est peut-être pas inutile de rappeler, avant tout autre exposé, quelques principes des mécanismes d'interconnexion de sous-réseaux.

2.1.1. Principes généraux d'interconnexion

Le but d'un tel type d'interconnexion est que *deux systèmes terminaux reliés à des sous-réseaux distincts doivent pouvoir communiquer et*

s'échanger des données qui seront transmises et ensuite comprises sans ambiguïté par les deux systèmes coopérants.[40] Nous envisagerons ici les problèmes d'interconnexion au sein du monde TCP/IP, protocoles qui seront examinés en détail plus avant. La structure en couches proposée par le modèle TCP/IP est la suivante :

Processus d'Application
Transport
IP (Interconnexion)
Accès au sous-réseau (couche optionnelle)
Liaison de données
Physique

Figure 4 : Structure en couches du modèle TCP/IP

Examinons le cas de deux terminaux distants interconnectés par un réseau. Ce réseau est constitué d'un ensemble de sous-réseaux reliés par des machines dites *passerelles* (*gateways* ou routeurs) supportant le même protocole d'interconnexion et qui assurent le transfert de données entre deux sous-réseaux. Les systèmes terminaux reliés à chaque sous-réseau, supportant également le protocole d'interconnexion, sont appelés *machines hôtes*.

L'objectif de l'interconnexion est de réaliser pour ces deux machines une liaison entre les sous-réseaux auxquels elles appartiennent respectivement et de fournir un service de compatibilité entre ceux-ci. Cela nécessite :

[40] Nachtergaele, V., *Interconnexion : présentation des principes généraux d'interconnexion*, avril 97, p.1, in Nachtergaele, V. et van Bastelaer, Ph., *Cours de compléments de téléinformatique, op. cit.*

- la prise en compte du routage des unités de données au sein du réseau, notamment au niveau des différents modes d'adressage au sein des sous-réseaux, des différentes tailles des unités de données, des différents mécanismes d'accès aux sous-réseaux, des traitements d'erreurs et des techniques de routage ;
- la fourniture de ces unités de données entre des processus sur des sous-réseaux distincts, tout en évitant les modifications d'architecture des sous-réseaux.

L'interconnexion de sous-réseaux peut être concrètement réalisée au niveau de différentes couches du modèle TCP/IP :

- au niveau Physique via des répéteurs ;
- au niveau de la Liaison de Données via des bridges ;
- au niveau IP via des routeurs ;
- au niveau Application via des passerelles.

Nous nous intéresserons à la troisième de ces possibilités : l'interconnexion « classique » au niveau de la couche IP du modèle TCP/IP.

2.1.2. Interconnexion au niveau de la couche IP

Reprenons à l'effet de cet exposé le texte donné par V. Nachtergaele au sujet de l'interconnexion dans le cadre du cours de compléments de téléinformatique de M. van Bastelaer :

> *La suite de protocoles TCP/IP a été conçue sur base de quatre concepts majeurs qui ont donné naissance à la structure en couches que nous connaissons. Il s'agit de la notion de* processus *qui s'exécutent sur des* machines hôtes *et qui communiquent entre eux en s'échangeant des données au travers de* sous-réseaux. *L'*interconnexion *de ces sous-réseaux est évidemment nécessaire afin de permettre l'échange de données de bout-en-bout entre les machines hôtes au travers d'un réseau (Internet) unique.*

La structure en quatre couches de TCP/IP est donc basée principalement sur la notion d'interconnexion (cf. figure 5). Cette structure repose sur le choix fondamental de limiter l'interconnexion au mode non connecté. En effet :

- *La couche la plus basse (couche d'Accès au réseau) implémente le protocole d'Accès au sous-réseau auquel la machine hôte accède. Elle supporte donc une série de protocoles permettant chacun l'accès à un type de sous-réseau donné* [...].
- *La couche IP (Internet Protocol) prend en charge la gestion de l'interconnexion en mode non connecté des différents sous-réseaux et les fonctions de routage et de relais des données de passerelle*[41] *à passerelle au travers du réseau Internet.*
- *La couche Transport supporte des protocoles de transfert de données de bout-en-bout entre les machines hôtes. Il s'agit des protocoles en mode connecté TCP (Transmission Control Protocol) et en mode non connecté UDP (User Datagram Protocol).*
- *Finalement, la couche supérieure implémente des processus d'applications offrant des services orientés vers l'application, tels que le transfert de fichiers, la messagerie électronique,* [...].[42]

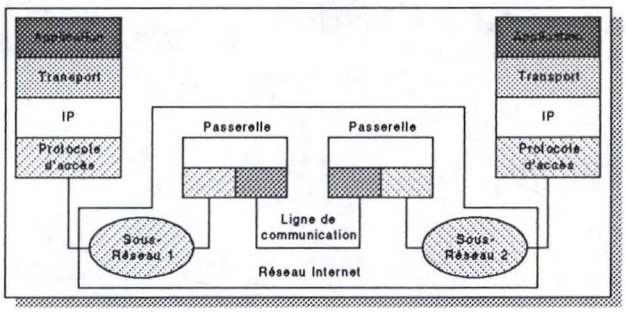


Figure 5 : Interconnexion TCP/IP

Après cet exposé sur les principes généraux d'interconnexion, nous pouvons passer à l'examen plus détaillé des différents protocoles que nous

[41] *Dans le monde TCP/IP, le terme «passerelle» (Gateway) est réservé pour désigner une entité interconnectant deux sous-réseaux au niveau de la couche IP. Une telle passerelle est aussi souvent appelée «routeur IP».*

[42] Nachtergaele, V., *Interconnexion : présentation des principes généraux d'interconnexion, op. cit.*, p. 7.

avons mentionné : IP et TCP, ainsi que d'un protocole plus particulièrement lié au Web : HTTP.

2.2. Protocoles[43]

2.2.1. IP

2.2.1.1. Introduction

IP (*Internet Protocol*) est l'un des *still points of a turning world* relevés lors d'une conférence[44] par Michel van den Bossche. Il s'agit du protocole d'Interconnexion le plus répandu. L'objectif d'IP, qui a été développé par le DoD (*Department of Defense*) des États-Unis et qui est défini par le document RFC 791, est d'assurer en mode non connecté les fonctions d'interconnexion de plusieurs sous-réseaux de manière transparente pour la couche supérieure (la couche Transport).

2.2.1.2. Caractéristiques d'IP

IP est donc un protocole en mode non connecté visant à transférer des données fournies par la couche Transport au sein d'unités de données dites *datagrammes*. IP se charge aussi de l'acheminement (le *routage*) des datagrammes à travers le réseau vers la machine hôte destinatrice et des fonctions d'interconnexion.

IP garantit que, lorsqu'un datagramme arrive, c'est à la bonne destination ; il garantit aussi l'identité de la machine émettrice. En revanche, IP ne garantit pas la transmission correcte ni le respect de l'ordre des datagrammes : les unités de données peuvent ne pas arriver à la machine de

---

[43] Cet exposé s'inspire de Tanenbaum, A., *Réseaux - Architectures, protocoles, applications*, InterEditions, Paris, 1990 et de Nachtergaele, V. et van Bastelaer, Ph., *Cours de compléments de téléinformatique*, *op. cit.*

[44] Van den Bossche, M., *Still points of a turning world*, conférence donnée à l'Institut d'informatique, Namur, mai 1998.

destination, arriver dans le désordre ou en plusieurs exemplaires, ou avec des erreurs dans la partie données du datagramme, puisque IP n'effectue pas de contrôle de détection des erreurs sur ce champ. Il sera du ressort de la ou des couche(s) supérieure(s) d'effectuer les contrôles et vérifications nécessaires pour assurer l'intégrité du transfert de données.

Le routage des datagrammes de machine en machine au sein du réseau se fait sur base de l'adresse de destination contenue dans chaque unité de données IP et des tables de routage dont disposent les machines passerelles.

IP assure aussi une fonction de fragmentation et de réassemblage des paquets, qui intervient lorsqu'un paquet est d'une taille trop grande pour le sous-réseau au travers duquel il doit transiter. IP fragmente alors ce paquet en plusieurs morceaux d'une taille compatible avec celle des unités de données de ce sous-réseau. L'ensemble des morceaux est envoyé sous forme de datagrammes à travers le sous-réseau, puis le paquet originel est reconstruit par l'entité IP sur la machine hôte réceptrice avant d'être transmis à l'entité Transport.

2.2.1.3. Adressage IP

Cf. annexe.

2.2.1.4. Primitives de service

Cf. annexe.

2.2.1.5. Format du datagramme IP

Cf. annexe.

2.2.2. TCP

2.2.2.1. Introduction

TCP (*Transmission Control Protocol*) est un protocole développé par le DoD (*Department of Defense*) et qui fait partie de la suite TCP/IP. Il est défini par le document RFC 793. Son rôle est de garantir le transfert correct des données, en mettant en œuvre des mécanismes qui suppléent aux carences du service offert par la couche d'interconnexion assuré par IP. Il s'agit de garantir que toutes les données arrivent au destinataire, sans erreur et en respectant l'ordre d'émission. TCP appartient à la couche Transport et offre un service en mode connecté (*connection oriented*) ; il utilise IP pour transmettre les données.

2.2.2.2. Caractéristiques de TCP

Le service offert par TCP est donc *connection oriented* et assure un transport fiable de bout en bout pour l'échange de données entre des processus sur des machines distinctes au travers d'un réseau Internet. Cinq points caractérisent ce service :

- *Quand deux (processus d') applications échangent des informations, TCP voit les données du processus d'application sous la forme d'un flot de données (octets). C'est à la couche TCP de diviser ce flot de données en unités de données (de protocole) de taille acceptable pour la transmission, appelées* ***segments****. Le processus d'application destinataire reçoit les données dans l'ordre dans lequel l'application source les a émises, mais pas nécessairement au même rythme.*
- *Le* ***flot de données*** *transmis entre deux processus d'application au moyen de TCP n'a* ***aucune structure significative pour le protocole TCP****. Il s'agit simplement d'une suite d'octets organisée en blocs. C'est de la responsabilité de l'application que de comprendre la signification de la suite d'octets.*
- *Quand une application désire communiquer avec une autre sur une machine distincte, elle demande au protocole TCP d'ouvrir une communication entre les deux machines (ouverture d'une connexion). Lorsque la connexion est établie, les applications peuvent alors échanger des données comme si un câble les reliait*

directement, de façon transparente, sans erreur, sans perte, sans duplication de données et dans le respect de l'ordre d'émission. TCP doit donc fournir un moyen de communication ***fiable****.*

- *Les processus d'applications soumettent leurs données à la couche TCP sous la forme d'un flot de données. TCP peut décider d'attendre d'avoir une quantité de données suffisante pour transmettre l'ensemble ou au contraire décider de transmettre les données par plus petits segments en fonction du réseau utilisé. Il existe cependant un mécanisme qui permet à une application de demander à l'entité TCP émettrice d'envoyer « sans attendre » les données soumises, et cela de manière telle que l'entité TCP réceptrice les transmette dès réception au processus d'application (« sans attendre » signifie que l'entité TCP émettrice ne peut attendre une quantité de données plus importante avant de transmettre l'ensemble). Un second mécanisme permet à l'application de soumettre des données à la couche TCP en signalant leur caractère urgent. Dans ce cas, ces* ***données « urgentes »*** *sont transmises par l'entité TCP émettrice avant les données déjà soumises à l'entité TCP et en attente d'une quantité plus importante pour transmission. De manière similaire, ces données « urgentes » seront délivrées à l'application par l'entité TCP réceptrice, avant les données déjà reçues mais non soumises au processus récepteur en attente d'une quantité suffisante.* [...]
- *La connexion offerte par TCP est* ***full-duplex****. Cela signifie que les deux processus d'application peuvent émettre simultanément des données.*[45]

Les principaux services offerts par TCP, qui seront détaillés plus avant, se composent des éléments suivants :

- Etablissement de la connexion ;
- Transfert des données ;
- Traitement des erreurs éventuelles (perte de données, non-respect de l'ordre d'émission des données) ;
- Fermeture de la connexion.

Outre ces services principaux, TCP dispose de certaines fonctions supplémentaires :

---

[45] Nachtergaele, V., *Couche TCP. Description d'une Couche Transport : La Couche TCP du DoD*, avril 97, pp. 2-3, in Nachtergaele, V. et van Bastelaer, Ph., *Cours de compléments de téléinformatique, op. cit.*

- Mécanisme de livraison de données urgentes ;
- Spécification d'une qualité de service désirée par l'utilisateur.

2.2.2.3. Identification des processus

Dans le monde TCP/IP, chaque processus est identifié par son *adresse*, appelée ***socket***. Un *socket* est formé de l'adresse IP de la machine hôte et d'un numéro de port identifiant le processus au sein de la machine. Le port est le point par lequel un processus d'application communique avec l'entité TCP.[46] Un *socket* est schématisé à la figure suivante :

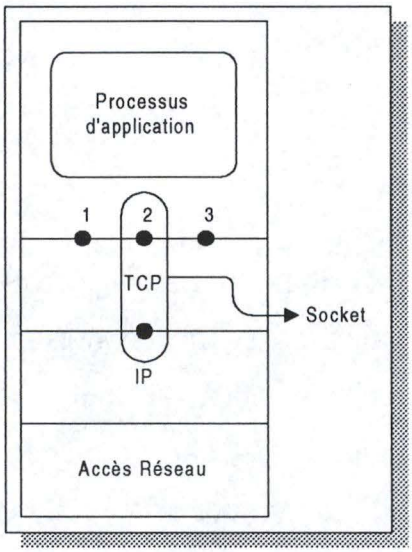


Figure 6 : Schéma d'un *socket*

2.2.2.4. Ouverture d'une connexion

a) Ouverture passive

Une entité TCP d'une machine hôte n'accepte l'établissement d'une connexion avec une autre entité TCP d'une autre machine hôte qu'à la condition d'avoir été préalablement mise en état, par le processus

[46] Un même processus peut être identifié au sein d'une machine par plusieurs ports ; il peut donc posséder plusieurs *sockets* au sein d'une même machine.

d'application correspondant, d'établir cette connexion par une demande explicite d'ouverture passive de connexion. Un processus d'application indique donc, à l'entité TCP de la machine sur laquelle il se trouve, qu'il est disposé à accepter l'ouverture d'une connexion qui lui serait destinée, par une demande d'ouverture passive de connexion à cette entité TCP qui, à ce moment, assigne à ce processus un port qui l'identifie.

Une ouverture passive peut être soit entièrement spécifiée (*fully specified passive open*) si un processus attend une demande d'ouverture active en provenance d'un processus particulier (défini par un *socket*), soit non spécifiée (*unspecified passive open*) s'il attend une demande d'ouverture active par un processus quelconque.

b) Ouverture active

A ce moment, aucune connexion n'est encore ouverte. Lorsqu'arrive une demande d'ouverture active de connexion, en provenance de l'entité TCP d'une autre machine hôte, si tout va bien, une connexion est normalement établie, identifiée par deux couples formés chacun d'une adresse IP et d'un port (*i.e.* la connexion est identifiée par les deux *sockets* qui constituent ses extrémités).

La figure suivante présente l'ouverture d'une connexion TCP, comprenant la phase d'ouverture passive et celle d'ouverture active :

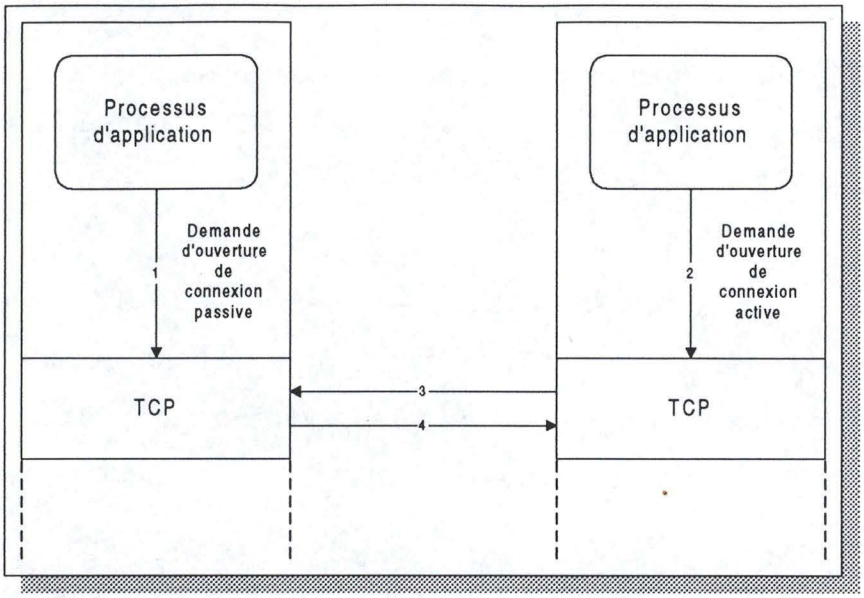


Figure 7 : Ouverture d'une connexion TCP

Il est à noter qu'entre deux *sockets* particuliers, il ne peut exister qu'une et une seule connexion TCP. Cependant, un même *socket* peut supporter plusieurs connexions simultanées, si chacune de ces connexions le mettent en rapport avec un *socket* distinct.

2.2.2.5. Transfert de données

a) Transfert de données standard

Quand l'ensemble de la procédure de connexion décrite ci-dessus a été réalisé par les processus situés sur les machines hôtes distinctes, ceux-ci peuvent commencer à échanger des données.[47] Comme nous l'avons dit plus haut, TCP divise le flot de données qu'il reçoit de la couche supérieure en segments : les unités de données de protocole ou PDU (*Protocol Data Unit*). Les PDU sont transmis tels quels à la couche d'Interconnexion assurée par IP.

[47] Nous considérons dans ce paragraphe l'échange normal de données. Le mécanisme de transfert des données urgentes est traité plus loin.

La répartition du flot de données en segments est le fait d'une opération de **segmentation**, qui consiste à encapsuler chaque morceau issu du flot de données dans un PDU et à le concaténer avec certaines informations de contrôle.[48]

A la réception, le pendant de la segmentation est le mécanisme de **réassemblage**, qui consiste à débarrasser les données des informations supplémentaires ajoutées lors de la segmentation et à fournir à la couche supérieure le flot de données tel qu'il avait été soumis à l'entité TCP émettrice.[49]

b) Mécanisme du transfert de données

Le protocole TCP garantit un service comportant les caractéristiques suivantes : la transmission des données de la machine hôte émettrice vers la machine hôte réceptrice se fait

- sans erreur ;
- sans dédoublement ;
- avec respect de l'ordre d'émission.

Pour offrir ce service, TCP doit prendre en charge les contrôles et vérifications nécessaires, puisque ceux-ci ne sont pas effectués par la couche inférieure assurée par IP. Le système utilisé par TCP est l'**acquittement positif avec retransmission** (*Positive Acknowledgement with Retransmission*) et est basé sur un principe d'accusé de réception fondé sur la technique de la **fenêtre coulissante** avec **piggybacking** (qui, simultanément, assure le contrôle de flux). Pour l'exposé des modalités du transfert, nous reprenons le texte de V. Nachtergaele :

[48] La segmentation n'est pas utilisée si la quantité de données soumise à l'entité TCP par le processus d'application est suffisamment peu élevée.
[49] Une différence peut apparaître au niveau du débit du flot de données.

L'entité TCP émettrice conserve les segments qu'elle a déjà envoyés et pour lesquels elle n'a pas encore reçu d'acquittement dans un buffer *appelé* ***fenêtre****. L'entité TCP émettrice peut ainsi transmettre les données reçues du processus d'application et en conserver une copie dans la fenêtre, tant qu'il reste des places disponibles dans cette dernière. Lorsque la fenêtre est remplie, l'entité TCP émettrice doit stopper la transmission jusqu'à l'arrivée d'une confirmation.*

Si aucune confirmation n'est reçue après un certain délai (appelé timeout*), l'entité réémet les données toujours présentes dans la fenêtre. Dès qu'une confirmation est reçue, l'entité TCP émettrice peut « déplacer » la fenêtre en y supprimant la (les) donnée(s) pour laquelle (lesquelles) la confirmation de réception vient d'être reçue (voir figure 11). Cette opération libère des positions dans la fenêtre, permettant à l'entité TCP émettrice de transmettre de nouveaux segments.*

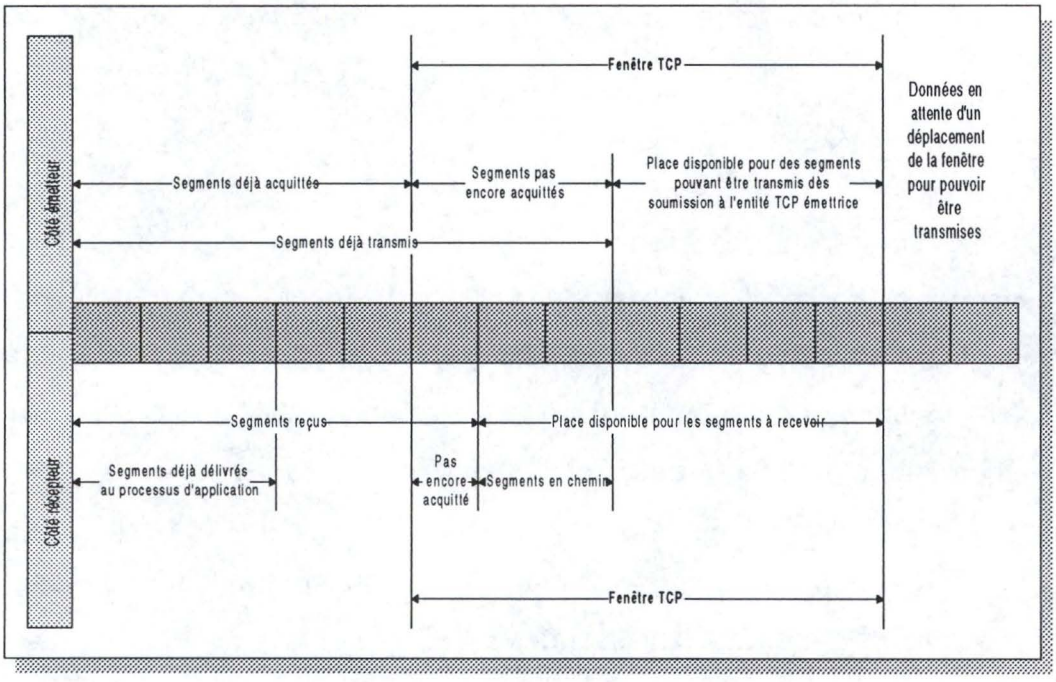


Figure 8 : Mécanisme d'acquittement et fenêtre

L'entité TCP réceptrice possède également une fenêtre contenant les segments déjà reçus mais non encore délivrés au processus d'application destinataire

Ce mécanisme d'acquittement avec fenêtre permet également de résoudre le problème du contrôle de flux des données, en permettant à l'entité TCP destinatrice d'arrêter momentanément la

transmission (par le simple fait de ne pas envoyer d'accusé de réception) jusqu'au moment où elle sera à nouveau prête à accepter de nouvelles données.

De plus, le mécanisme de fenêtre utilisé dans le protocole TCP permet à l'entité TCP réceptrice de modifier la taille de la fenêtre de l'entité TCP émettrice, afin (par exemple) de forcer une transmission plus lente des segments. Lorsqu'elle est totalement submergée par l'arrivée de segments, l'entité TCP réceptrice peut momentanément forcer une taille de fenêtre nulle chez l'entité TCP émettrice, afin de stopper complètement la transmission des données. Dès qu'elle est à nouveau prête à accepter de nouveaux segments, elle rétablira une taille de fenêtre jugée acceptable, permettant ainsi la reprise de la transmission des données.

Notons encore que, la connexion TCP se faisant en mode full-duplex, *les deux entités TCP sont simultanément émettrices et réceptrices. Chaque entité possède donc deux fenêtres, une fenêtre d'émission des segments soumis par le processus d'application et l'autre consacrée aux segments reçus de l'entité homologue.*[50]

c) Transfert de données urgentes

Lorsqu'un processus d'application demande une transmission de données urgentes à son entité TCP, celle-ci génère immédiatement un segment ne contenant que ces données et des informations de contrôle, qui est transmis prioritairement à travers le réseau, c'est-à-dire qu'aucun autre segment ne peut être transmis avant celui-ci.

La transmission de données urgentes se fait donc en dehors du mécanisme de fenêtre coulissante.

2.2.2.6. Primitives de service

Cf. annexe.

[50] Nachtergaele, V., *Couche TCP. Description d'une Couche Transport : La Couche TCP du DoD, op. cit.*, pp. 8-9

2.2.2.7. Format du segment TCP

Cf. annexe.

Après avoir examiné dans le détail le fonctionnement des couches Interconnexion (avec le protocole IP) et Transport (avec le protocole TCP), nous pouvons passer à l'exposé des caractéristiques d'un des protocoles les plus utilisés sur le *Web* : HTTP.

2.2.3. HTTP

2.2.3.1. Introduction

Nous avons vu, plus haut, que l'intérêt principal d'Internet résidait dans la possibilité, pour un utilisateur, de consulter des documents. Un document est, au sein du réseau, identifié par son *URL* (*Uniform Resource Locator*), qui est constitué du nom d'un protocole (par exemple FTP ou HTTP), du nom de domaine (*Domain Name*) ou de l'adresse IP de la machine hôte sur laquelle se trouve le serveur du document, éventuellement d'un numéro de port TCP de ce serveur, et du chemin d'accès (*path*) de ce document au sein de la structure de fichiers de ce serveur.

L'utilisateur accède à un document par un appel à son URL via un *browser*, qui formalise la requête selon le protocole spécifié dans l'URL. Cette requête est ensuite véhiculée au travers du réseau jusqu'au serveur qui dispose du document. Ce document est alors transmis au client via une réponse selon le même protocole qu'à l'aller.

Nous envisagerons ici le cas où le protocole utilisé est HTTP (*Hyper Text Transfer Protocol*), protocole développé pour le *Web* et qui, comme son nom l'indique, est spécialisé dans la transmission de documents hypermedia. Ceux-ci sont généralement rédigés selon un langage propre : HTML (*Hyper*

Text Markup Language). Plus d'informations sont données plus loin au sujet de ce langage ; nous nous intéressons d'abord à HTTP.

2.2.3.2. Caractéristiques de HTTP

Le protocole HTTP se situe comme illustré sur la figure suivante au sein de l'architecture générale du *Web* :

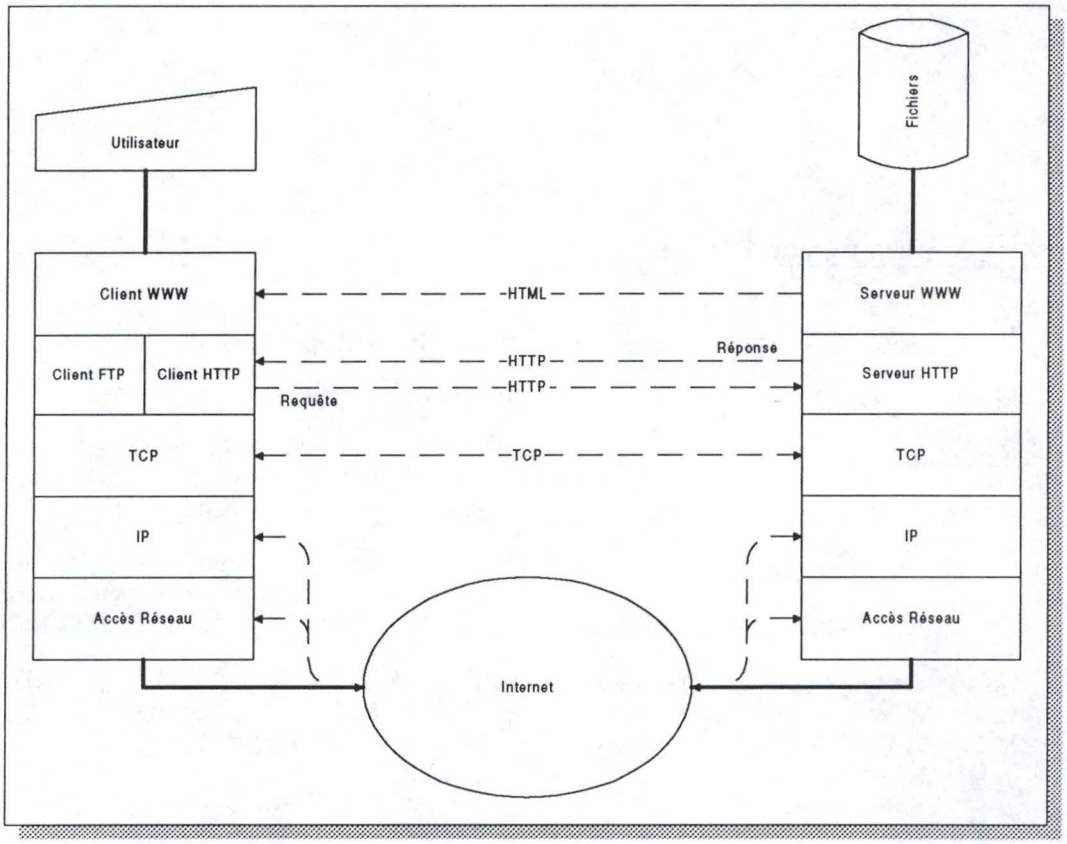


Figure 9 : Architecture générale du WWW

Comme on peut le voir sur la figure, l'architecture générale du WWW repose sur un modèle Client/Serveur. Au sein de ce modèle, un client WWW utilise un client HTTP qui dialogue avec un serveur HTTP lui-même utilisé par un serveur WWW. Par ailleurs, le protocole HTTP utilise les services de la suite TCP/IP pour la transmission des documents (cas général, non interactif) et offre un service de formulaire qui permet le déclenchement d'un script sur le serveur (cas interactif, *cf. infra*).

2.2.3.3. Primitives de service

Cf. annexe.

2.2.3.4. Format du PDU HTTP

Cf. annexe.

2.3. Langages

Dans cette section, nous évoquons deux langages fréquemment associés avec le monde du *World-Wide Web*. De nombreux autres le sont aussi, mais nous avons mis l'accent sur les outils que nous utiliserons dans la troisième partie de ce mémoire.

Le premier de ces langages est **HTML** (*Hyper Text Markup Language*), qui n'est pas à proprement parler un langage de programmation, mais un langage servant à décrire les documents publiés sur le *Web*.

Le second de ces langages est **Java** qui, lui, est un véritable langage de programmation, offrant des possibilités semblables à celles de C++.

2.3.1. HTML[51]

2.3.1.1. Un langage issu de SGML

Lors de sa conception, il a été décidé que le *Web* devait être pourvu d'une interface commune et simple d'utilisation. A cette fin a été développé le

Hyper Text Markup Language (HTML), constitué à partir d'un sous-ensemble du *Standard Generalized Markup Language* (SGML). Comme son nom l'indique, SGML est un langage normalisé de balisage généralisé, dont l'objectif est de séparer la forme, la structure logique et le contenu de documents au moyen de balises. Les avantages de l'utilisation de la norme SGML sont les suivants :

- Indépendance par rapport aux matériels, échange facilité des textes ;
- Séparation de la structure logique et de la mise en forme des documents ;
- Indépendance par rapport aux logiciels, donnant les avantages des systèmes ouverts pour le traitement des données ;
- Format sans ambiguïté, permettant la mémorisation et la restauration dans et à partir des bases de données.

SGML rend les documents portables parce qu'il permet de décrire sans ambiguïtés leurs structures logiques, indépendamment de leur forme visuelle. SGML n'est pas spécifique à un fournisseur et est contrôlé par l'ISO (*International Standard Association*). Celle-ci crée et modifie les normes dans un processus ouvert au public par l'intermédiaire de participation aux groupes nationaux de normalisation. SGML est donc une norme ouverte.

SGML est utilisé pour définir des langages permettant de décrire des documents. A ce titre, SGML est suffisamment flexible pour définir un ensemble infini de langages de balisage générique. Un langage de balisage SGML définit les structures hiérarchiques possibles d'un document dans sa classe. La définition d'un langage de balisage par SGML est appelée « création d'une application SGML » ou « création d'une **définition de type de document** » (*Document Type Definition - DTD*). HTML n'est rien d'autre qu'une telle DTD.

---

[51] Cet exposé s'inspire de : Stanek, R. S., *HTML, JAVA, CGI, VRML, SGML - Secrets d'experts*, Simon & Schuster Macmillan, Paris, 1996, et de Van Herwijnen, E., *SGML Pratique*, International Thomson Publishing France, Paris, 1995.

2.3.1.2. Caractéristiques et possibilités de HTML[52]

Seuls les éléments essentiels de SGML ont été retenus pour former la spécification initiale de HTML. Cela a eu pour effet de réduire sensiblement la complexité de la spécification initiale et, par suite, de réduire les temps de transfert des hyperdocuments sur le réseau. Un autre avantage de l'utilisation de SGML comme base de HTML est que les définitions de type de document de SGML fournissent un moyen simple d'étendre la norme HTML.

Cette possibilité n'a pas été oubliée et la spécification de HTML a beaucoup changé depuis sa naissance. Chaque nouvelle spécification est cependant parfaitement compatible avec les précédentes et comporte de nombreuses améliorations à la norme. En plus de ces spécifications, beaucoup d'éditeurs *Web* utilisent des extensions à la norme, qui correspondent à des solutions pour certains problèmes d'édition *Web* que les spécifications actuelles ne savent pas traiter.

a) HTML 1.0

HTML 1.0 est la spécification initiale du *Hyper Text Markup language*. A cause des possibilités limitées de cette spécification, les documents créés avec HTML 1.0 sont d'une conception assez rudimentaire. On y trouve :

- Plusieurs niveaux de titre ;
- Les paragraphes ;
- Les références hypertexte ;
- Un formatage spécial pour les listes.

[52] Nous ne mentionnons ici que les possibilités générales offertes par HTML pour la publication de documents sur le Web. Pour une définition détaillée des balises et de leur utilisation, nous renvoyons le lecteur intéressé à l'abondante documentation disponible sur Internet.

b) HTML 2.0

HTML 2.0 permet un contrôle plus fin sur la présentation et la mise en valeur du texte dans les documents *Web*. Il offre en outre :

- Les images (le début des possibilités multimédia de HTML) ;
- Les formulaires à remplir (permettant l'interactivité).

c) HTML 3.0

Avant même la ratification de HTML 2.0, les éditeurs *Web*, désireux de créer des documents encore meilleurs et plus impressionnants, ont commencé à prendre en considération les fonctionnalités avancées de HTML 3.0, telles que :

- Un contrôle accru de la présentation ;
- Les bannières ;
- La gestion par le client des zones actives des images ;
- Des listes personnalisées ;
- Des documents dynamiques ;
- Des formules mathématiques ;
- Des feuilles de style ;
- Des tables (et des tables dans les formulaires).

HTML est en amélioration constante. D'autres fonctionnalités sont déjà prévues pour la norme HTML 4.0 ou DHTML. Comme la plupart sont encore en cours de spécification, nous ne nous étendrons pas sur ces nouvelles possibilités. Examinons simplement encore les extensions offertes par les *browsers* :

d) Extensions HTML de Netscape

Nestcape Communications Corporation a réalisé les extensions les plus populaires à la norme HTML, et cela au fil des différentes versions de son navigateur :

- Compléments concernant la présentation (centrage, clignotement, *etc.*) ;
- Extensions de la règle horizontale pour la largeur, la longueur et les hachures ;
- Contrôle sur les tailles relatives des polices de caractères ;
- Contrôle sur la couleur des caractères ;
- Utilisation des images ou des couleurs pour définir le fond d'un document ;
- Possibilité pour un document d'avoir plusieurs fenêtres (*frames*) ;
- Langage de script de Netscape pour des scripts gérés par le client ;
- Possibilité d'intégrer des objets multimédia et d'utiliser un module additionnel (*plug-in*) dans le *browser.*

e) Extensions HTML de *Internet Explorer*

Le *browser* développé par Microsoft fut le premier à reconnaître directement le multimédia en interne :

- Spécification des types et des couleurs des caractères ;
- Utilisation de messages défilants ;
- Utilisation de sources vidéo ou audio dynamiques afin d'avoir des films vidéo en ligne ;
- Création de documents munis de bandes son.

f) Extensions HTML pour Java

Avant de parler du langage Java proprement dit, il est bon de mentionner que HTML permet l'intégration, au sein de documents *Web*, de véritables applications en ligne appelées *applets* (*cf. infra*).

L'incorporation des programmes écrits en Java dans les pages HTML se fait par l'intermédiaire de la balise <applet>.

2.3.2. Java[53]

2.3.2.1. Introduction : Java répond à un besoin

Au fil du temps, et malgré la relative interactivité apportée par les formulaires que permet HTML, s'est fait sentir le besoin d'une plus grande communication interactive entre le *Web* et ses utilisateurs. Cela a nécessité de revoir la façon de concevoir les applications Client/Serveur qui, jusqu'alors, n'étaient gérées que de manière très approximative. Un grand nombre de développeurs de sites *Web* ramenaient la technologie Client/Serveur à un client qui accède aux données qui se trouvent sur le serveur. Or, dans un rapport Client/Serveur véritable, les données devraient être transmises dans les deux sens entre les machines client et serveur, sans qu'aucune des deux ne fasse la totalité du travail à elle seule. Les tâches nécessitant la manipulation de beaucoup de données et prenant beaucoup de temps devraient être exécutées sur le serveur, qui est d'ordinaire une machine performante et qui a pleinement accès aux bases de données. Un PC en local peut traiter les tâches plus simples. Si le client réalise une partie du traitement, cela réduit le trafic entre l'ordinateur de l'utilisateur et les serveurs sur Internet, ce qui a pour effet d'accélérer les accès. De plus,

[53] Cet exposé s'inspire de : December, J., Morrison, M. *et al.*, *JAVA - Secrets d'experts*, Simon & Schuster Macmillan, Paris, 1996, de Lemay, L., Perkins, Ch. L., *Apprenez Java 1.1 en 21 jours*, Simon & Schuster Macmillan, coll. Le Programmeur, Paris, 1997 et de Stanek, R. S., *HTML, JAVA, CGI, VRML, SGML - Secrets d'experts*, Simon & Schuster Macmillan, Paris, 1996.

puisque beaucoup de tâches sont exécutées sur un ordinateur local, la vitesse globale de tout le site *Web* semble accrue.

Cette philosophie est pleinement rencontrée par le langage Java, développé par *Sun Microsystems*, dont les applications ont pour caractéristique principale d'être portables : leur code (source et binaire) est compatible avec différents ordinateurs. Le langage Java permet le développement d'*applets*, qui sont de petits programmes destinés à transiter, *via* le réseau, du serveur vers le client et à s'exécuter sur celui-ci, rééquilibrant de cette manière le rapport Client/Serveur tel qu'il a été évoqué ci-dessus..

2.3.2.2. Caractéristiques générales de Java

Java, en tant que langage de programmation, est caractérisé par les traits suivants : il s'agit d'un langage **orienté objet**, **distribué**, **interprété**, relativement **typé**, préoccupé des aspects relatifs à la **sécurité**, **indépendant** de l'**architecture matérielle**, **portable**, relativement **performant**, **multithread** et **dynamique**. Toutes ces particularités méritent bien chacune un petit mot d'explication :

a) Java est orienté objet

Java permet une approche orientée objet de l'écriture de logiciels. La programmation objet s'appuie sur la modélisation du monde en termes de composants logiciels appelés objets. Un **objet** se compose de **données** et d'opérations, ou **méthodes**, qui peuvent être effectuées sur ces données. Ces méthodes peuvent **encapsuler**, c'est-à-dire protéger, les données d'un objet car les programmeurs peuvent créer des objets dans lesquels les méthodes constituent le seul moyen de vérifier l'état des données.

Une autre caractéristique de l'approche objet, supportée par Java, est la possibilité d'**héritage**. Les objets peuvent utiliser des caractéristiques d'autres objets sans avoir à dupliquer les fonctionnalités de ces derniers objets. L'héritage contribue donc, comme l'encapsulation, à la **réutilisation** du logiciel et de ses modules, car les programmeurs n'ont plus qu'à créer les méthodes chargées d'effectuer une tâche donnée, une fois et une seule.

Un autre avantage de l'héritage concerne l'organisation du logiciel, l'architecture logicielle et sa lisibilité. En rassemblant les objets dans des **classes**, chaque objet d'une classe va hériter des caractéristiques des objets parents, plus généraux. Cela facilite le travail de documentation, de compréhension et de récupération des versions précédentes du logiciel. En effet, l'ensemble des fonctionnalités du logiciel se développe progressivement à mesure que de nouveaux objets sont créés.

b) Java est distribué

Java a été conçu spécifiquement pour fonctionner dans des environnements de **réseaux interconnectés**. Java possède donc une volumineuse bibliothèque de classes pour la communication via les **protocoles Internet** basés sur TCP/IP, notamment HTTP et FTP. Les programmes Java peuvent notamment manipuler aisément des ressources par le biais des URL.

c) Java est interprété

Ce qui peut passer à première vue pour un défaut se révèle ici être une qualité. Le compilateur Java ne transforme pas le fichier source en code binaire, mais en un **code intermédiaire** qui, lui, est interprété sur ce qu'on appelle une *Java Virtual Machine*. Celle-ci transforme ce code intermédiaire en code binaire exécutable sur l'ordinateur particulier sur lequel elle tourne.

Ce mécanisme est la clef de la **portabilité**, puisque le code intermédiaire peut être exécuté sur toute machine disposant d'un interprète Java ou d'un *browser* compatible Java. Cela permet d'écrire les programmes Java indépendamment des plates-formes matérielles dont disposent les utilisateurs.

L'on pourrait s'interroger sur les performances d'un code intermédiaire Java interprété. Celles-ci ne sont pas toujours aussi bonnes que celles obtenues par compilation directe et exécution sur une plate-forme matérielle donnée. Le compilateur de l'environnement Java comporte une option pour traduire le code intermédiaire en code machine spécifique à une plate-forme donnée. Il est possible d'obtenir ainsi des performances comparables au processus habituel de compilation et de chargement en mémoire.

d) Java est typé

En Java, le typage est relativement rigoureux. Si le langage ne demande pas une déclaration des types des variables aussi poussée qu'en Pascal, par exemple, il n'est toutefois pas possible pour un programmeur de transtyper un entier quelconque en pointeur sans autre forme de procès.

Par ailleurs, Java n'accepte pas l'arithmétique sur les pointeurs et ne permet pas à un programmeur d'écraser des zones mémoire, ni d'altérer des données au moyen de pointeurs. Ces caractéristiques participent aussi de la sécurisation offerte par Java (*cf. infra*).

e) Java est sécurisé

Comme Java fonctionne dans des environnements de réseaux interconnectés, la question de la sécurité se pose avec acuité. Le langage comprend des restrictions sévères sur les accès à la mémoire : interdiction

de l'**arithmétique sur les pointeurs** et de l'emploi d'opérateurs de **transtypage** aberrants. En outre, les interprètes Java sont tous munis d'une **routine de vérification** du code intermédiaire qui s'assure que les instructions de ce code ne violent aucune des règles du langage, qu'elles ne créent pas de faux pointeurs, n'accèdent pas à une zone de mémoire interdite ou à des objets en dehors de ce que permet leur définition. Cette routine vérifie aussi que les appels de méthode comportent le nombre correct d'arguments, que ces arguments sont du bon type et qu'il ne se produit aucun débordement de pile. Une vérification des noms de classe et des conditions d'accès a lieu pendant le chargement. Il existe aussi un système de sécurité par interfaces qui permet de mettre en œuvre des mesures de sécurité à divers niveaux.

Au niveau de l'accès aux fichiers, si une instruction du code intermédiaire tente d'accéder à un fichier pour lequel elle n'a pas d'autorisation, une boîte de dialogue sera affichée pour demander à l'utilisateur s'il veut continuer ou terminer l'exécution du programme. Au niveau du réseau, les versions futures auront des possibilités de mettre en œuvre un cryptage par clef publique ainsi que d'autres techniques cryptographiques pour vérifier l'origine du code et son intégrité une fois qu'il aura traversé le réseau. A l'exécution, il est possible d'utiliser des informations sur la provenance du code intermédiaire pour décider ce que ce code a le droit de faire.

f) Java est multithread

Java permet de créer des applications capables d'effectuer « plusieurs actions en même temps ». Grâce à un ensemble de routines permettant de suivre plusieurs « *threads* » (ou successions d'événements au fil du temps) et basées sur le paradigme des moniteurs dû à C.A.R. Hoare[54], Java propose

[54] *Cf.*, par exemple, Hoare, C.A.R., *Monitors : An Operating System Structuring Concept*, Comm. ACM 17, 1994.

aux programmeurs un moyen de réaliser dans leurs programmes un comportement interactif et dynamique en temps réel.

3. Conclusion de la deuxième partie

Internet est une réalité vaste qui ne se limite pas, nous l'avons dit, au *World-Wide Web*. Si celui-ci est avant tout un concept, il s'est imposé comme le moyen d'interaction standard au sein du Réseau des réseaux. Par ailleurs, si l'origine d'Internet est d'abord militaire, puis universitaire, ce nouveau média est aujourd'hui populaire, tant au niveau des particuliers qu'à celui des entreprises.

Après avoir esquissé deux des tendances, le commerce électronique et la technologie *Push*, qui nous semblaient s'inscrire adéquatement dans la perspective de notre mémoire et qui achevaient, avec une réflexion sur l'opportunité de mettre Internet en relation avec les IN, le volet consacré au « pourquoi » Internet, nous nous sommes intéressé à une dimension plus technique du Réseau des réseaux. Il a fallu d'abord reprendre quelques principes généraux d'interconnexion afin de pouvoir aisément passer à la description de trois protocoles fréquemment rencontrés sur Internet : IP, TCP et HTTP. Rappelons que seules les caractéristiques générales de ces protocoles ont été données dans le corps du mémoire ; – des détails complémentaires sont fournis en annexe. Une évocation a ensuite été donnée de deux langages souvent associés à Internet : un langage de description, HTML, de la norme duquel on a marqué les grandes évolutions, et un langage de programmation, Java, dont on a indiqué les traits particuliers.

Il est à présent temps de passer à la partie de ce mémoire relative à l'exposé de l'implémentation de notre application de monitoring.

Partie III. Monitoring d'un IN via le *Web*

0. Introduction : définition du problème

Après avoir examiné les éléments constituants et les caractéristiques des Réseaux Intelligents et d'Internet, nous allons nous appliquer à les mettre en correspondance afin d'atteindre le but que nous poursuivons : implémenter le prototype d'une application de monitoring de Réseau Intelligent via Internet. Cernons tout d'abord le problème.

Le monitoring d'un Réseau Intelligent se présente avant tout comme une activité de surveillance. De ce point de vue, on peut considérer un Réseau Intelligent comme un générateur d'événements. Les événements dont il est question sont des descriptions d'alarmes en provenance du Réseau ; le monitoring consiste à prendre connaissance de ces descriptions d'alarmes et de les examiner (éventuellement en vue d'une action sur le Réseau lui-même).

L'on voudrait trouver un moyen permettant d'effectuer une activité de ce type par le biais d'Internet. Plus précisément, la question était d'abord de savoir si, parmi les technologies de type *Push* disponibles sur Internet, il était possible d'en trouver une qui fût susceptible d'apporter une aide au monitoring de Réseau Intelligent de façon fiable (sûre) et efficace (rapide). Dans le cas où aucune de ces technologies ne serait à même de répondre à nos besoins, il faudrait se résoudre à développer un prototype de solution indépendant de ces techniques. Cette solution devrait prendre en compte les faits suivants : l'information relative au monitoring serait localisée sur un site accessible *via* Internet, l'utilisateur du monitoring se trouverait devant un PC connecté au Réseau, l'information doit, à certains moments déterminés, être envoyée (« poussée ») par ce site vers le PC de l'utilisateur.

1. Choix de solutions

Nous présentons dans cette section un aperçu de deux technologies qui, à première vue, semblent remplir les objectifs que nous nous sommes fixés pour répondre à notre problème. Ces technologies sont ensuite critiquées dans la perspective de ce problème ; nous déterminons enfin si elles méritent d'être retenues ou non pour la solution que nous mettrons en œuvre.

1.1. Éventail de possibilités

Plusieurs manières de résoudre notre problème se présentent immédiatement à l'esprit. Parmi elles, la technologie *Push* proprement dite, telle qu'elle est proposée (sous différentes formes) par diverses sociétés sur Internet, et le courrier électronique. Examinons-les plus en détail.

1.1.1. Le Push

Le *Push* est une technologie relativement récente qui, en toute généralité, repose théoriquement sur le principe suivant : il s'agit non plus d'aller *chercher* l'information là où elle se trouve sur le *Web*, mais de faire en sorte qu'elle soit *reçue* par l'utilisateur qui en a fait la demande, au moment et sous la forme qu'il désire, sans autre intervention de sa part. Nous sommes donc face à une autre philosophie de l'interaction sur Internet.

Le *Push* est utilisé principalement dans les domaines de l'information et des loisirs, et repose communément sur le principe suivant. Des sociétés commerciales créent, sur Internet, des *channels* (canaux) diffusant de l'information de façon thématique (par exemple : des informations boursières,

sportives, politiques ou culturelles).[55] Ces *channels* sont accessibles aux clients de ces sociétés par l'intermédiaire d'un abonnement. Lorsqu'il s'abonne, l'utilisateur qui désire bénéficier d'un service de ce type reçoit un logiciel qui permet la réception, la configuration et la consultation des *channels*. Ce logiciel fonctionne de la manière suivante : résidant sur la machine de l'utilisateur, il se connecte périodiquement (selon une fréquence déterminée lors de l'abonnement) aux *channels* définis par l'utilisateur et effectue une *mise à jour* des informations disponibles sur la base de celles qui ont été collectées lors de la connexion précédente. Une fois cette mise à jour achevée, il ferme la connexion jusqu'à la prochaine échéance de rafraîchissement. Dans la plupart des cas, la présentation des informations à l'utilisateur reste à sa guise : il peut choisir de les visualiser par l'intermédiaire de son *browser* ou ancrer une visionneuse sur son bureau virtuel (*webtop*).

1.1.2. Le courrier électronique

Une solution alternative à la technologie *Push* pourrait résider dans l'utilisation du courrier électronique. Dans le cas du monitoring, en effet, l'e-mail semble une solution élégante *a priori*. Plus précisément, il s'agirait de la *génération automatique de messages électroniques*. L'idée serait qu'à une fréquence déterminée, ou à la survenance d'un événement important dont il s'agira de déterminer les caractéristiques, un e-mail reprenant l'ensemble des alarmes correspondantes serait envoyé à la personne qui assure le monitoring. On voit bien les avantages de l'e-mail dans le cas qui nous occupe :

- *Indépendance* : par rapport aux *browsers*, aux plates-formes, aux applications qui les produisent et/ou les utilisent, *etc* ;
- *Confidentialité* : possibilité de crypter les messages ;

[55] Dans certains cas (par exemple : pour *Netscape NetCaster*), des *channels* peuvent être créés par des tiers.

- *Légèreté* : de développement, d'implémentation, d'utilisation, *etc.*

L'e-mail dispose encore d'un atout supplémentaire : le *paging*. Il s'agit de la possibilité de recevoir (le cas échéant, d'émettre) de courts messages (en général alphanumériques) sur un *pager* ou un GSM équipé de cette fonctionnalité. Nous entrevoyons ici deux modalités de l'utilisation du *paging* :

- notification *via* le *pager* (ou l'écran du GSM) de l'arrivée d'un e-mail ;
- visualisation directe du message.

1.2. Critique des possibilités

1.2.1. Le Push

Du fait de la diffusion des logiciels de *Push* et des *channels* eux-mêmes par des sociétés commerciales, il va de soi que celles-ci utilisent des technologies propriétaires : les *channels* proposés aux utilisateurs de *NetCaster* ne le sont pas, par exemple, aux utilisateurs de *PointCast*. En outre, si l'on veut utiliser son *browser* (*Internet Explorer* de *Microsoft*, *Navigator* ou *Communicator* de *Netscape*, *etc.*) pour visualiser les informations des *channels* choisis, il faut s'assurer qu'il y a compatibilité entre ces différents logiciels.

Une fois cette première critique envisagée vient la seconde, la plus importante. Elle tient au fonctionnement même de la technologie *Push*. En réalité, l'information n'est pas, comme l'on pourrait s'y attendre, *poussée* vers la machine de l'utilisateur, mais bien, de façon traditionnelle, *tirée* de l'endroit où elle se trouve sur le *Web* vers cette même machine. La seule différence réside dans le fait que ce n'est plus l'utilisateur qui s'occupe de cette tâche, mais un programme spécialement dédié à celle-ci, résidant sur sa machine, et configuré de façon à répondre aux attentes de l'utilisateur. L'on peut aussi

faire remarquer que l'usage de la technologie *Push* s'apparente davantage à la *mise à jour* d'informations existantes qu'à la recherche *active* de nouvelles informations (comme le feraient, par exemple, des agents intelligents), et qu'elle occasionne une charge de travail non négligeable, tant pour la machine de l'utilisateur que pour le réseau lui-même.

1.2.2. Le courrier électronique

La génération automatique de courrier électronique, bien qu'elle réponde mieux à la définition de la technologie *Push*, - puisqu'il n'y a ici aucune intervention de l'utilisateur à qui est destinée l'information, sauf à déterminer sa fréquence -, souffre malheureusement de défauts rédhibitoires.

Le premier de ces défauts se rapporte aux retards qu'un e-mail est susceptible de subir, pour différentes causes, lors de sa transmission entre la source et la destination. Le second défaut tient à la probabilité, faible mais non nulle, qu'un message se perde et n'arrive jamais à destination. Ces inconvénients, qui l'éliminent de la course à la solution, ne doivent pas nous faire oublier l'intéressante possibilité qu'offre l'e-mail : le *paging*.

1.3. Conclusion

Pour toutes ces raisons, et dans l'objectif de trouver une solution adéquate au problème qui nous occupe, tant la technologie *Push*, telle qu'elle est proposée par les sociétés qui la promeuvent, que la génération automatique de messages électroniques ne semblent répondre à nos besoins. Dès lors, vers quoi s'orienter ?

Il semble qu'une solution conjoignant les qualités d'indépendance, d'efficacité, de fiabilité et de sécurité que l'on attend d'elle doive être définie outre l'offre du marché. Cette solution, intuitivement, serait basée sur une architecture Client / Serveur qui reste à définir.

2. Implémentation

Nous présentons dans cette section les différentes étapes qui ont mené à la réalisation de notre prototype d'application de monitoring d'un Réseau Intelligent via Internet. Ces étapes correspondent d'abord aux rencontres avec les membres du groupe Alcatel : une première approche, avec une définition large du problème et des solutions à y apporter, a été suivie d'une seconde approche, plus restreinte quant à ses ambitions, eu égard, notamment, au fait que ce mémoire, à l'origine prévu pour deux étudiants, ne sera finalement réalisé que par un seul.

Les autres étapes se rapportent ensuite à la découpe classique des phases de l'implémentation d'un projet informatique.

2.1. Approches successives

2.1.1. Première approche (large)

2.1.1.1. Description générale

Selon notre première rencontre avec les membres d'Alcatel, le 14/11/1997, l'architecture Client / Serveur de notre application de monitoring de Réseau Intelligent aurait comporté des modules écrits en Java 1.1 et en C++.[56] Elle aurait correspondu *grosso modo* à une utilisation fondée sur le scénario suivant :

1. l'utilisateur désirant mener le monitoring se connecte par l'intermédiaire de sa machine (Client) au Serveur qui lui soumet un formulaire de souscription ;

[56] Pour des raisons de compatibilité matérielle.

2. l'utilisateur complète le formulaire en y introduisant les données relatives au type de monitoring qu'il souhaite effectuer et renvoie le formulaire complété au Serveur ;
3. si le Serveur accepte la souscription de l'utilisateur, il donne à celui-ci le moyen de recevoir de sa part les alarmes en provenance du Réseau Intelligent dont il s'agit de faire le monitoring, alarmes qui correspondent aux données du formulaire que l'utilisateur a complété ;
4. le monitoring s'achève conformément à ces mêmes données.

2.1.1.2. Découpe en modules

Lorsqu'on examine la description générale de notre application, on peut y discerner cinq entités jouant chacune un rôle différent dans le scénario exposé ci-dessus. Ces cinq entités sont les suivantes :

1. une entité se chargeant du formulaire à envoyer à l'utilisateur, qui doit le compléter ;
2. une entité gérant la souscription de l'utilisateur au monitoring ;
3. une entité s'occupant de collecter les descriptions d'alarmes en provenance du Réseau Intelligent ;
4. une entité traitant l'envoi vers l'utilisateur des descriptions d'alarmes collectées correspondant aux données qu'il a indiquées dans le formulaire ;
5. une entité veillant à la réception et à la mise en forme pour l'utilisateur des descriptions d'alarmes qui lui ont été envoyées.

En termes de programmation, ces cinq entités auraient donné lieu à cinq modules distincts, qui seraient tous stockés sur le Serveur, mais s'exécuteraient, selon leurs fonctionnalités, soit sur le Serveur, soit sur le Client. Ces cinq modules sont les suivants (leurs fonctionnalités correspondent à celles des entités décrites ci-dessus) :

1. un module *Form*, écrit en HTML ou en Java 1.1 (à décider), qui s'exécute sur le Client ;
2. un module *Subscript*, écrit en C++, qui s'exécute sur le Serveur ;
3. un module *Collect*, écrit en C++, qui s'exécute sur le Serveur ;
4. un module *Send*, écrit en C++, qui s'exécute sur le Serveur ;
5. un module *Listener*, écrit en Java 1.1, qui s'exécute sur le Client.

La communication entre les modules se fait par l'intermédiaire de *sockets*, via le protocole TCP tel que nous l'avons exposé *supra*. Seuls les modules *Form* (*applet* Java ou formulaire HTML) et *Listener* (*applet* Java) disposent d'une interface utilisateur graphique.

On peut schématiser l'architecture générale du système de la façon suivante :

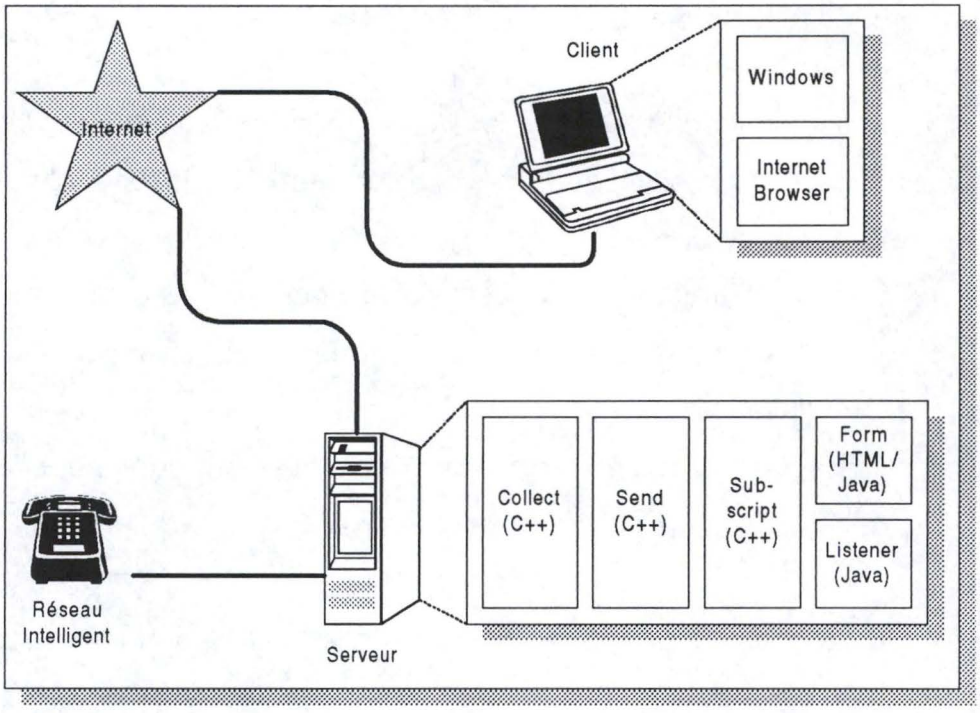


Figure 10 : Architecture du système de monitoring (1^e approche)

2.1.1.3. Description des modules

a) Le module *Form*

Le module *Form* s'occupe de recueillir les données concernant le monitoring en provenance de l'utilisateur et de les transmettre au Serveur.

Lorsqu'un utilisateur désire effectuer le monitoring d'un Réseau Intelligent, il lance sur sa machine (Client) une session de son *browser* Internet habituel (par exemple *Navigator* de *Netscape* ou *Internet Explorer* de *Microsoft*), et se connecte au site adéquat, celui du Serveur (SMP - *Service Management Point*). Le *browser* rapatrie sur le Client la page HTML correspondante, qui contient un formulaire (sous forme d'*applet* ou de page HTML, à décider) dont les champs, au minimum, sont les suivants :

- identification de l'utilisateur ;
- mot de passe ;
- type de monitoring souhaité (*on line* et/ou *e-mail*) ;
- niveau et type d'alarme à partir duquel l'utilisateur désire être informé[57] ;
- fréquence de rafraîchissement des informations souhaitée ;
- durée souhaitée du monitoring.

L'utilisateur complète ensuite les différents champs, - ils sont tous obligatoires mais peuvent comporter des valeurs par défaut -, et valide sa requête.

Si le formulaire a été transmis à l'utilisateur sous la forme d'une *applet*, celle-ci, sur base des champs d'identification et de mot de passe, est alors en mesure de déterminer, selon une méthode qui reste à préciser, si l'utilisateur est en droit de procéder au monitoring.

---

[57] On suppose les alarmes classées par niveaux et types d'importance catégorisée.

Si le formulaire a été transmis à l'utilisateur sous la forme d'une page HTML, un clic sur le bouton *Send* envoie les données du formulaire au Serveur (requête HTTP) qui effectue lui-même la vérification.[58]

Quelle qu'en soit la manière, si l'utilisateur, après contrôle, est autorisé à effectuer le monitoring, les données correspondant aux divers champs du formulaire sont acheminées vers le module *Subscript* sur le Serveur, et une (nouvelle) *applet*, *Listener*, est chargée à partir du Serveur vers la machine Client de l'utilisateur.

Au cas où l'utilisateur ne serait pas autorisé à effectuer le monitoring, on peut imaginer qu'une deuxième, puis une troisième chance soient données à l'utilisateur, avant que toute possibilité de souscription au monitoring lui soit retirée.

b) Le module *Subscript*

Lorsque la requête provenant de l'utilisateur est validée, les données correspondant aux différents champs du formulaire de souscription au monitoring sont envoyées au module *Subscript*. On peut imaginer que celui-ci les range dans un fichier correspondant à une table conservée en mémoire centrale sur le Serveur.

[58] Reste à s'assurer de la sécurité et de la confidentialité de la transmission de ces données sensibles via HTTP.

c) Le module *Collect*

Le module *Collect* est chargé de recevoir les descriptions d'alarmes en provenance du Réseau Intelligent dont il s'agit d'assurer le monitoring. On peut imaginer qu'il les range dans un fichier correspondant à une table conservée, si possible, en mémoire centrale, sinon sur disque.

d) Le module *Send*

Lorsque vient à échéance le moment de rafraîchir les informations selon la fréquence définie par l'utilisateur, ou lorsqu'un événement doit être signalé sans attendre, le module *Send* se charge de mettre en relation les données de la table du module *Subscript* avec celles de la table du module *Collect*. Cette mise en relation pourrait se faire, par exemple, par le biais d'une requête SQL opérant une jointure dont les arguments sont les deux tables précitées et dont le critère de sélection est l'égalité champ à champ des caractéristiques du monitoring désiré par l'utilisateur.

Une nouvelle table est ainsi créée, qui est envoyée, en une fois, vers la machine de l'utilisateur. Si cette table est vide, un message affirmant la poursuite du monitoring est envoyé à sa place.

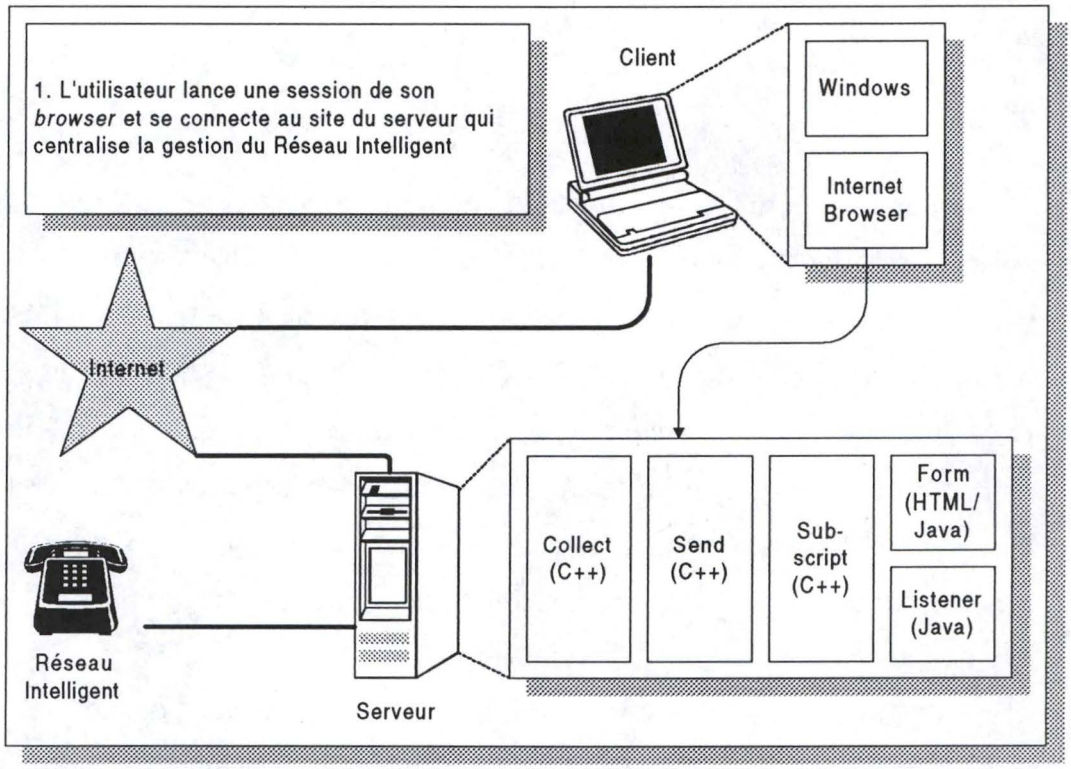
e) Le module *Listener*

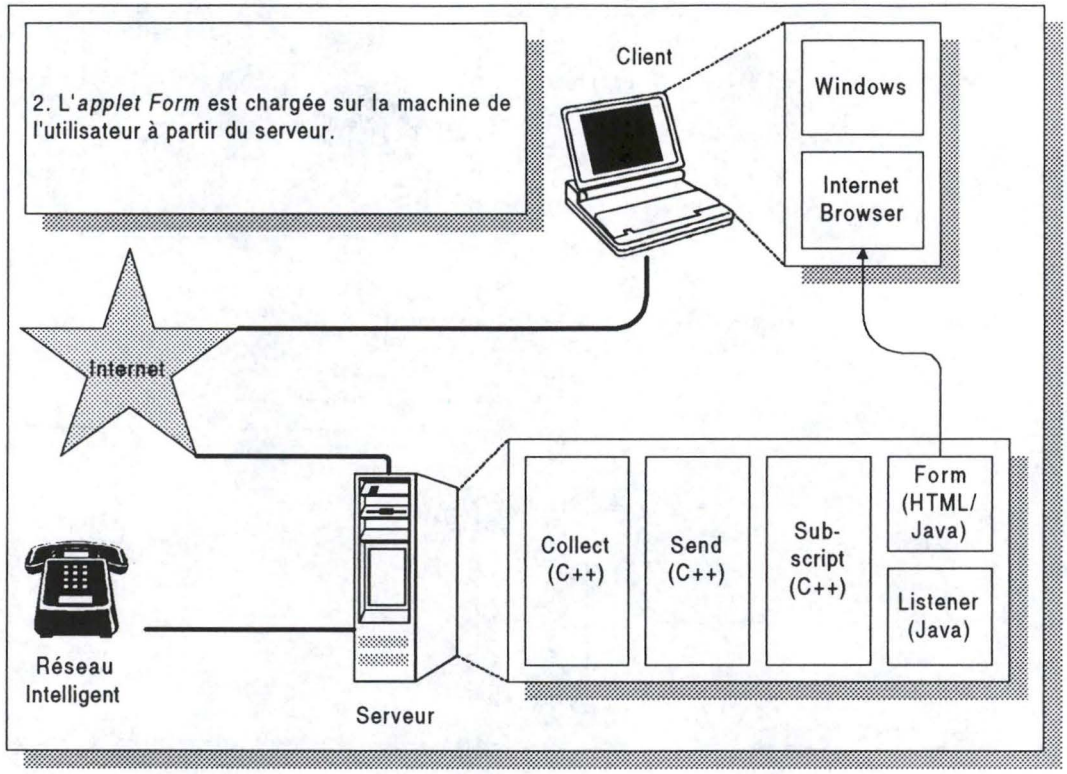
L'*applet Listener* succède au formulaire sur la machine de l'utilisateur, dans le cas où celui-ci a été autorisé à effectuer le monitoring. Cette *applet* est chargée de rester à l'écoute d'une éventuelle transmission de données par le module *Send*.

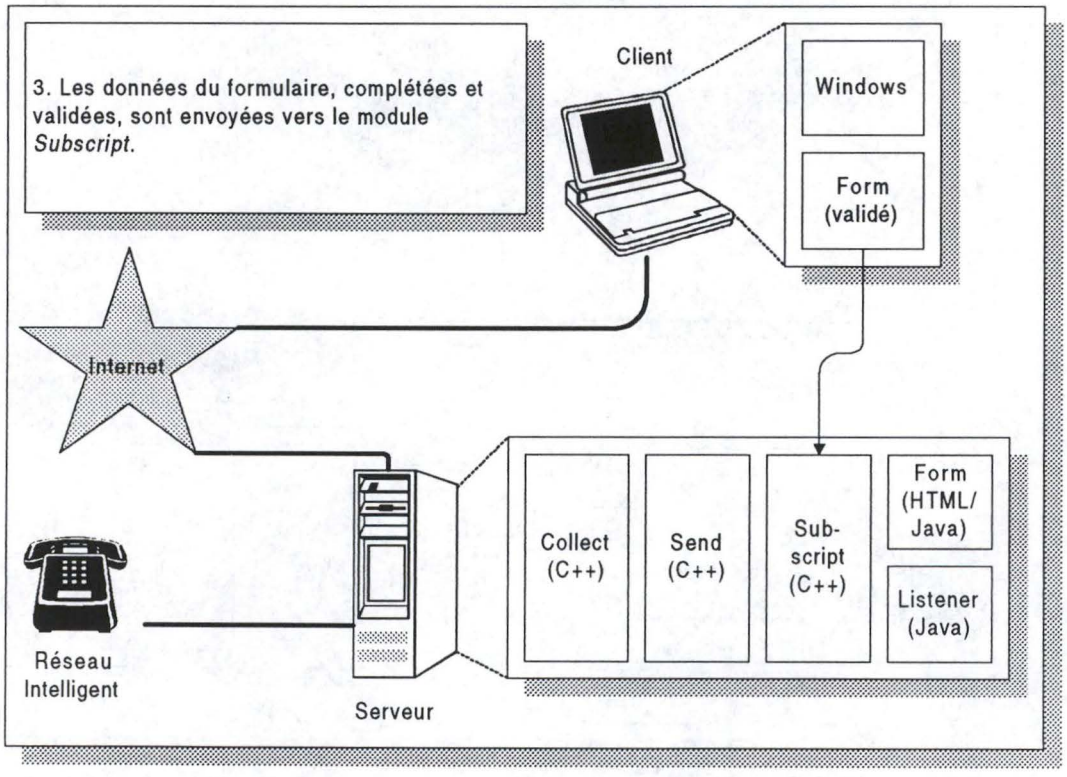
Lorsqu'arrive une table de données décrivant les alarmes du Réseau Intelligent, l'*applet Listener* met ces données sous une forme visualisable par

l'utilisateur. S'il s'agit d'un message avertissant simplement de la poursuite du monitoring, elle se contente de l'afficher.

On peut sommairement schématiser le scénario d'utilisation de la façon suivante :




2. L'*applet Form* est chargée sur la machine de l'utilisateur à partir du serveur.
Client
Windows
Internet Browser
Internet
Collect (C++)
Send (C++)
Sub-script (C++)
Form (HTML/ Java)
Listener (Java)
Réseau Intelligent
Serveur


3. Les données du formulaire, complétées et validées, sont envoyées vers le module *Subscript*.
Client
Windows
Form (validé)
Internet
Collect (C++)
Send (C++)
Sub-script (C++)
Form (HTML/ Java)
Listener (Java)
Réseau Intelligent
Serveur

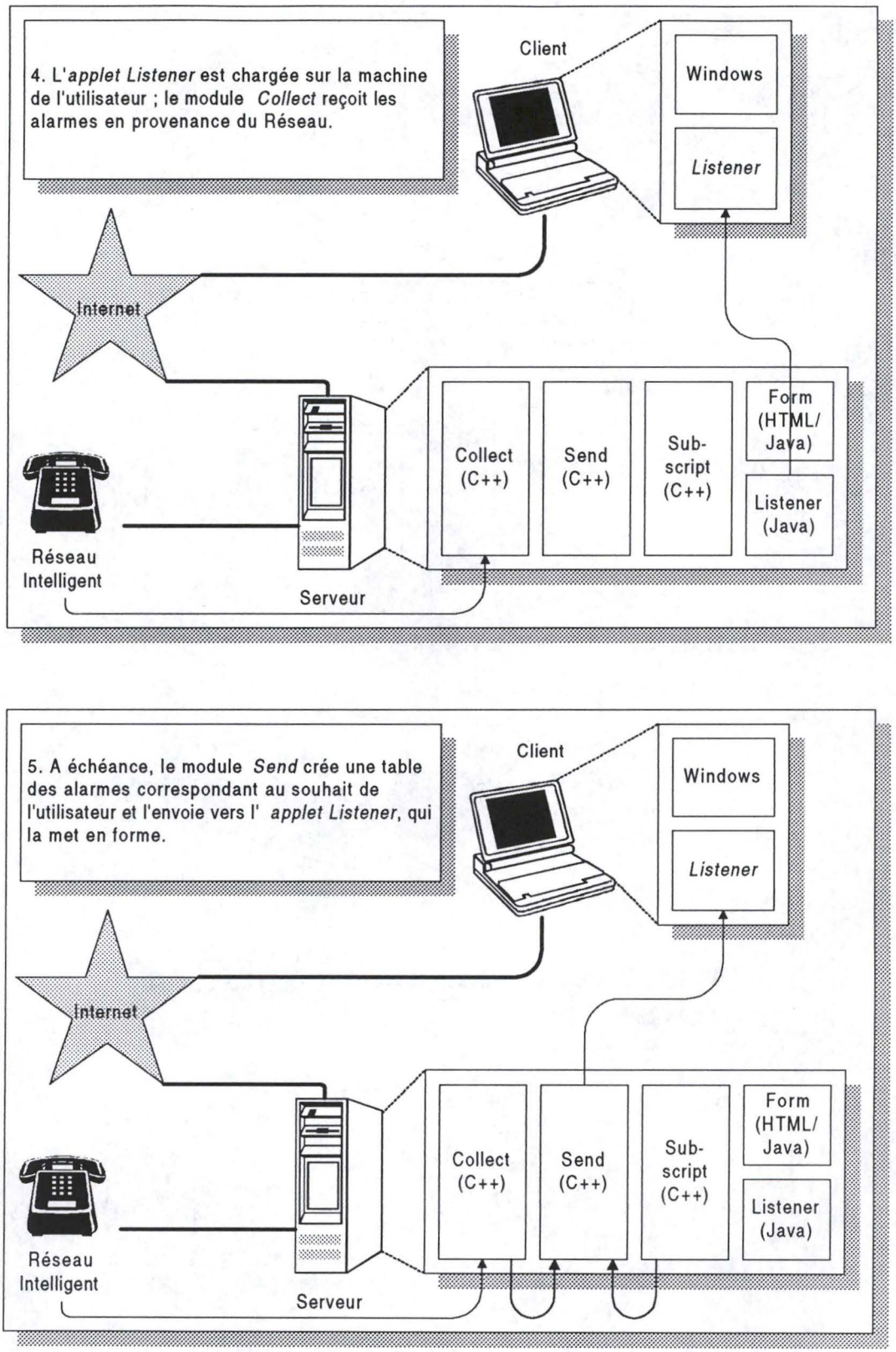


Figure 11 : Scénario de fonctionnement du monitoring (1e approche)

2.1.2. Seconde approche (restreinte)

Il a été décidé d'apporter plusieurs simplifications au premier projet présenté *supra*. Ces simplifications tiennent compte, nous l'avons dit, du fait que ce mémoire était originellement prévu pour deux étudiants ; elles se justifient en outre chacune par d'autres éléments.

Tout d'abord, il a été jugé que l'apprentissage d'un seul langage de programmation nouveau, dans le cadre du mémoire, était suffisant. L'on s'en tiendra donc à Java.

Ensuite, l'attention s'est focalisée sur l'aspect Client/Serveur de l'application à développer. Comme celui-ci est le modèle de l'ensemble des transactions sur Internet, le développement de l'application « en local » ne posera pas le problème de la transposition à l'échelle du *Web*.

Comme il s'agit d'un prototype, l'accent a été mis sur la transparence des fonctionnalités du logiciel. C'est pourquoi l'interface graphique qui, développée au moyen d'outils de quatrième génération d'aide à la programmation tels que *Symantec Cafe* ou *Borland J-Builder*, « cache » une grande partie du code nécessaire à l'implémentation par le recours à de nombreux éléments de bibliothèques fournies avec ces logiciels, l'interface graphique, disons-nous, a été abandonnée au profit de la ligne de commande qui, pour être moins agréable esthétiquement, a le mérite de présenter clairement et de laisser aisément modifier, le cas échéant,[59] les arguments nécessaires à l'exécution des fonctionnalités du logiciel.

Dans la même perspective, le monitoring est réalisé sous forme d'application Java et non d'*applet*, dans la mesure où l'on évite ainsi les restrictions sévères liées à la sécurité et imposées par Java à ses *applets*, tout en conservant la possibilité aisée de transformer l'application en *applet*.

---

[59] Rappelons qu'il s'agit ici d'un prototype.

Voilà les principales simplifications qui ont été apportées au projet. Le cœur en est néanmoins resté par la conservation de l'architecture Client/Serveur, de la communication entre ces deux entités reposant sur la suite TCP/IP, et de la majeure partie des fonctionnalités attendues du logiciel.

2.2. Prototype

2.2.0. Introduction

Ci-dessous sont données les notes relatives à l'implémentation de notre application de monitoring de Réseau Intelligent. Ces notes suivent les étapes qui ont jalonné le chemin menant à l'aboutissement du projet.

Les explications qu'elles contiennent sont assez succinctes, d'une part parce que la décomposition en étapes rend celles-ci relativement simples, d'autre part parce que nous croyons avoir fourni une assez bonne documentation dans le code lui-même, que nous avons placé en annexe. Nous avons aussi annexé des captures d'écran de logiciels en fonctionnement.

2.2.1. Développement d'un générateur d'événements

La base de notre implémentation repose, tout d'abord, sur une modélisation adéquate d'un Réseau Intelligent. Comme nous l'avons dit, dans la perspective d'un prototype d'application de surveillance, l'IN dont il s'agit d'assurer le monitoring peut être considéré comme un générateur d'événements. Les événements pris en compte dans ce cadre sont des alarmes : on suppose que certaines actions effectuées sur, par ou au moyen du Réseau entraînent l'apparition de situations susceptibles de devoir être

communiquées (leur occurrence comme leur description) dans un certain délai à certaines personnes, qui pourront ou devront alors effectuer certaines actions en rapport avec la survenance de ces situations.

Nous modéliserons donc un Réseau Intelligent par un générateur d'alarmes aléatoires. Une alarme sera elle-même modélisée par une structure de données comportant au moins les champs suivants :

- Un numéro d'alarme ;
- Un type d'alarme (de 1 à 15) ;
- Un niveau d'alarme (de 1 à 10) ;
- Une date de survenance (au format jj/MM/aa hh:mm:ss) ;

La première orientation que nous avons suivie a été de créer une classe Java de générateur d'alarmes. Celle-ci s'appelait *Generateur* et ne comportait qu'une seule méthode, *main()*, qui prenait comme argument (issu de la ligne de commande), le nombre d'alarmes que l'utilisateur désirait faire générer. Après avoir vérifié la validité de cet argument, la méthode générait, au sein d'une boucle *for* dont l'indice était lié à l'argument, des alarmes caractérisées par un type et un niveau choisis au hasard sur base du générateur de nombres aléatoires fourni par la classe Random du package java.util. Ces alarmes étaient ensuite à la fois affichées à l'écran et écrites dans un fichier intitulé *alarmes.txt*.

Cette première approche n'était pas satisfaisante en termes de « réutilisabilité ». La seconde orientation que nous avons suivie s'inscrivait beaucoup mieux dans la plus pure tradition des techniques orientées objet enseignées à l'Institut. Elle passait par la définition non plus d'une classe *Generateur*, mais d'une classe *Alarme*, qui comportait les variables d'instance *numero*, *niveau*, *type* et *date*, et les méthodes *générerAlarme()*, *afficherAlarme()* et *main()*. Cette classe remplissait exactement les mêmes fonctions que la classe *Generateur*, mais on était en droit d'espérer

beaucoup plus d'elle en termes de « réutilisabilité ». Nous verrons ce qu'il en a été par la suite.

2.2.2. Ajout d'un filtre au générateur d'événements

La seconde étape fut constituée par l'adjonction à la classe Alarme d'une méthode qui permît de filtrer les alarmes, c'est-à-dire de les caractériser sur base de leur type et de leur niveau. L'implémentation de cette fonctionnalité nécessitait quelques modifications de la classe Alarme elle-même. Nous avons opté pour l'ajout, aux variables d'instance qui contenaient déjà le numéro, le niveau, le type et la date des alarmes, de deux indicateurs signalant l'un si l'alarme était filtrée, l'autre si l'alarme était prioritaire.[60]

La fonction de filtrage est le fait de la méthode *filtrerAlarme()* qui, sur base de quatre arguments supplémentaires fournis par l'utilisateur, représentant, les deux premiers, le niveau et le type minimaux d'alarme, les deux suivants, le niveau et le type minimaux prioritaires à partir desquels l'utilisateur désire être prévenu, marque les alarmes comme *filtrées* et/ou *prioritaires*. Le filtrage se fait de la façon suivante : si le niveau *et* le type de l'alarme considérée sont inférieurs aux niveau et type minimaux spécifiés par l'utilisateur, l'alarme est *filtrée* ; si le niveau *ou* le type de l'alarme considérée est inférieur au niveau ou au type minimal prioritaire spécifié par l'utilisateur, l'alarme est *prioritaire*.

La méthode *afficherAlarme()* subit elle aussi une modification visant à faire apparaître à l'écran si, lorsqu'elle est générée, une alarme est *filtrée* et/ou *prioritaire*. Si c'est le cas, l'alarme, en plus d'être consignée dans le fichier *alarmes.txt*, l'est aussi dans le fichier *resultats.txt*.

---

[60] Une alarme *filtrée* est communiquée à l'utilisateur à l'échéance du monitoring indiquée par sa fréquence, une alarme *prioritaire* est envoyée sans délai à l'utilisateur, c'est-à-dire même en dehors des échéances de monitoring.

2.2.3. Développement d'une architecture Client/Serveur

La troisième étape de notre parcours consista à élaborer la structure de communication entre le serveur qui disposerait des alarmes « générées par le Réseau » et l'utilisateur. Nous avons choisi de conserver la base de l'architecture Client/Serveur exposée plus haut. Le langage Java dispose de nombreuses facilités pour la programmation réseau, notamment les classes contenues dans le paquetage java.net.

La seule fonction que nous avons décidé d'implémenter afin de tester cette architecture est une fonction « perroquet » : le serveur se contente de répéter à l'utilisateur du client ce que celui-ci a entré au clavier à destination du serveur.

2.2.3.1. Le serveur

La classe que nous avons choisi d'appeler sobrement *Serveur* étend la classe *Thread*. A ce titre, elle redéfinit la méthode *run()* qui sera commentée plus loin.

La classe *Serveur* comporte une variable entière à laquelle correspond un numéro de port pour la connexion TCP entre le serveur et le client, et une variable de type ServerSocket qui représente cette connexion du côté du serveur. Elle dispose d'une méthode constructeur *Serveur()* qui appelle le constructeur de la classe parent (*Thread*) puis essaie de créer le socket serveur. La métohde *main()* crée une nouvelle instance de la classe *Serveur* puis la fait démarrer.

La méthode *run()* teste si le socket côté serveur a été créé, puis renvoie sous forme de socket une tentative acceptée d'un client qui souhaite se connecter au serveur. Elle définit ensuite des flux d'entrée et de sortie pour l'exécution des fonctionnalités. Un message de bienvenue est affiché du côté du client quand celui-ci a vu sa demande de connexion acceptée par le

serveur. Les entrées au clavier de l'utilisateur du client sont renvoyées par le serveur vers le client, jusqu'au moment où celui-ci entre la chaîne de caractères « Salut », qui achève à la fois l'exécution du client et celle du serveur.

2.2.3.2. Le client

La classe *Client* ne possède qu'une méthode, *main()*, qui déclare et initialise un socket qui représentera la connexion avec le serveur, et deux flux, l'un pour les entrées, l'autre pour les sorties. Cette méthode prend pour argument (issu de la ligne de commande) l'adresse du serveur (sous forme d'adresse IP ou de nom). L'échange entre le serveur et le client se fait alors au moyen de deux boucles *while* imbriquées, l'une attendant la fin de l'émission des données issues du serveur pour les afficher sur l'écran du client, l'autre attendant que l'utilisateur du client frappe la touche entrée pour envoyer les caractères saisis au serveur, qui les renverra vers le client (fonction « perroquet »). Ces caractères sont aussi affichés à l'écran du côté du serveur.

2.2.4. Intégration du générateur d'événements au serveur

L'intégration du générateur d'événements au Serveur a demandé des modifications tant de la classe *Serveur* que de la classe *Client*, même si l'architecture générale du système a été conservée. Ces modifications sont exposées ci-après.

2.2.4.1. Le serveur

Les méthodes constructeur (*Serveur()*) et *main()* sont les mêmes que celles qui ont été examinées *supra*. L'attente et la connexion d'un client sont elles aussi gérées de manière identique. Une fois un client connecté au serveur débute un dialogue qui vise d'abord à identifier et authentifier le client, puis à acquérir les paramètres qui seront nécessaires à l'exécution du

monitoring. Cette acquisition est le fait de la méthode *dialogue()*, qui elle-même fait appel à la méthode *vérification()* qui, comme son nom l'indique, vérifie la validité des arguments fournis par l'utilisateur du client en vue de l'exécution du monitoring. Celui-ci démarre ensuite. Il fait l'objet de la méthode *monitoring()*. Cette méthode fonctionne de manière analogue à la méthode *main()* de la classe *Alarme*. Une instance de cette classe est créée et des appels sont faits aux méthodes *générerAlarme()*, *filtrerAlarme()* et *afficherAlarme()* de cette classe. Ces appels sont effectués au sein d'une boucle *for* qui compte le nombre de cycles de monitoring à faire. Cette boucle *for* contient elle-même une boucle *while* qui assure la génération, le filtrage et l'affichage côté serveur des alarmes. Les alarmes prioritaires sont envoyées sans délai côté client. La boucle *while* se termine à l'échéance d'un cycle de monitoring (cette échéance est calculée sur la base de la fréquence et de l'heure du début du monitoring d'une part, de l'heure système d'autre part). A ce moment, les alarmes filtrées, dont les caractéristiques ont été écrites dans des tableaux réservés à cet usage, sont envoyées au client par la méthode *envoyerAlarmes()*. Côté serveur, les alarmes d'un cycle de monitoring sont écrites à chaque fin de cycle dans un fichier ayant un nom différent du fichier dans lequel sont écrites les alarmes du cycle précédent.

2.2.4.2. Le client

Le client conserve plus de caractéristiques avec la version précédente que le serveur. Seul le mode de dialogue a été adapté afin que, lorsque le monitoring a commencé (c'est-à-dire après l'appel à la méthode *monitoring()* à l'exécution du serveur), le client puisse afficher plusieurs lignes d'affilée. Une autre modification a été apportée pour permettre au client d'écrire les alarmes filtrées en provenance du serveur dans un fichier appelé resultats.txt. Ce fichier reprend donc l'ensemble des alarmes filtrées envoyées par le serveur pour toute la durée du monitoring. A la fin de celui-ci, l'utilisateur du client tape *exit* afin de stopper à la fois l'exécution du client et celle du serveur.

3. Conclusion de la troisième partie

La troisième étape du parcours que constitue ce mémoire s'intéressait à l'implémentation d'un prototype d'application de monitoring de Réseau Intelligent *via* Internet. Dans ce cadre, la première chose à considérer était la modélisation de l'IN dont il s'agissait d'assurer le monitoring : ce réseau était vu comme un générateur d'événements, c'est-à-dire d'alarmes. Nous sommes ensuite parti de la constatation que les technologies disponibles que nous avons examinées, *Push* et courrier électronique, ne répondaient pas aux caractéristiques de notre problème. Il nous a dès lors fallu nous orienter vers une solution « sur mesure ».

Nous ne nous étendrons pas sur la première approche de cette solution qui, trop ambitieuse, n'a pas été développée. Retenons simplement qu'elle était basée sur cinq modules (*Form*, *Subscript*, *Collect*, *Send* et *Listener*) et nécessitait un interfaçage Java - C++.

La seconde approche de cette solution est celle qui a été développée. Rappelons brièvement que cette approche met l'accent sur l'architecture Client/Serveur, modèle des transactions sur le Web, qu'elle abandonne l'interface graphique pour la ligne de commande, qu'elle prend la forme d'une application et non plus d'une *applet*. Le développement de cette application se compose des étapes suivantes : développement d'un générateur d'événements, ajout d'un filtre au générateur d'événements, développement d'une architecture Client/Serveur, intégration du générateur d'événements au serveur. Le code relatif à ces étapes successives est donné en annexe, ainsi que quelques captures d'écran sélectionnées lors de l'exécution du logiciel.

Le développement de ce prototype achevé, nous pouvons, en guise de conclusion à cette troisième partie, le critiquer. Il est évident que, dans la mesure où il ne s'agit que d'un prototype, l'application n'est pas prête à fonctionner telle quelle. Tout d'abord, elle n'a pas été complètement testée :

si elle semble s'exécuter correctement (notamment lors des tests qui ont permis les captures d'écran), elle n'a jamais été développée et testée que sur un seul PC, qui simulait une liaison réseau par l'exécution de deux fenêtres DOS. Ensuite, un grand nombre de choses sont à améliorer, voire à remplacer complètement : la vérification de l'identité et de la validité du mot de passe (qui, par ailleurs, devrait être masqué) fournis par l'utilisateur du client est tout ce qu'il y a de plus rudimentaire, la gestion du dialogue entre le client et le serveur, tout comme le code dans son ensemble, pourraient être grandement optimisés, des mécanismes de reprise après erreur (par exemple si l'utilisateur du client s'est trompé en donnant son mot de passe) devraient être inclus, *etc.* En outre, le prototype souffre d'un défaut de modélisation qui réside dans le fait que, comme c'est le serveur (*mono-thread*) qui génère les alarmes, celles-ci ne sont plus produites pendant que le serveur communique au client les alarmes qui ont été filtrées ... En réalité, la génération des alarmes ne devrait pas cesser.

Malgré tous ces défauts, et dans le cadre des simplifications que nous avons mentionnées lorsque nous sommes passé de la première à la seconde approche de la solution à implémenter, nous pensons que le prototype développé n'est pas inutile, dans la mesure où il est susceptible de donner les grandes lignes de l'architecture d'une application de monitoring. Même si ce prototype n'a pas été entièrement testé, il a le mérite de fonctionner correctement dans les conditions dans lesquelles il a été développé, de répondre pour une grand part aux besoins qui avaient été définis[61] et ainsi de donner une idée des échanges (et de la façon de les traiter) qui peuvent avoir lieu entre un client et un serveur dont l'objectif commun est d'assurer la surveillance d'un Réseau Intelligent.

[61] Pour rappel : 1. information relative au monitoring localisée sur un site accessible *via* Internet, 2. utilisateur du monitoring devant un PC connecté au Réseau, 3. information, à certains moments déterminés, envoyée (« poussée ») par ce site vers le PC de l'utilisateur.

Conclusion générale

Nous voici arrivé au terme du parcours. A tout le moins, celui-ci a été riche d'enseignements. Des récapitulations de la matière des sujets abordés ont été données à la fin de chacune des trois parties qui le composent, – c'est pourquoi nous préférons à présent mettre l'accent sur les apports de l'expérience constituée par la rédaction de ce mémoire.

Cette expérience est à considérer sous un double aspect. Le premier aspect est celui de la *connaissance*. Nous avons en effet appris à connaître un domaine, celui des Réseaux Intelligents, qui, bien qu'à la croisée des chemins de l'informatique et des télécommunications, comme nous l'avons vu, n'en était pas moins nouveau pour nous. A ce titre, il fut intéressant de ne pas considérer simplement cette notion comme une réalité ayant une existence déjà donnée, dont il eût suffi d'examiner les caractéristiques techniques, mais comme un produit à concevoir, à développer, à vendre et à implémenter. Pour cela, la prise en compte des Acteurs intervenant dans le monde des IN et du contexte de l'évolution de ces Réseaux n'était pas dénuée d'utilité. Bien entendu, l'exposé a également fait la part belle aux architectures, tant des Réseaux que des services, tant du matériel que du logiciel, – le thème du mémoire l'imposait. Ce thème nous a, en outre, permis d'approfondir certaines notions découvertes durant notre *cursus*, notamment en matière de protocoles, et d'en examiner d'autres, nouvelles, comme la technologie *Push*.

La connaissance étant peu de chose sans la *pratique*, c'est ce second aspect de notre expérience, peut-être le plus riche, que nous voudrions à présent évoquer. La mise en œuvre de l'apprentissage du langage Java en est la première et la plus grande part. La rédaction du mémoire proprement

dite en est la seconde. Le langage Java n'est pas foncièrement difficile à apprendre ni à mettre en œuvre, mais sa particularité réside dans le mode d'accession aux ressources qu'il offre, et auquel il faut se familiariser. En effet, si des livres existent (*cf.* bibliographie), l'immense majorité des ressources est accessible *on-line*. Cela est dû au fait que, Java étant un langage de pointe en perpétuel développement, le rythme de l'apparition de nouvelles classes, de classes modifiées ou même d'une nouvelle version du langage dépasse les possibilités des éditeurs de livres mais nécessite néanmoins le maintien d'une documentation accessible au plus grand nombre des développeurs Java. Cette tâche est admirablement accomplie par les promoteurs du langage, notamment sur les sites Internet que nous avons mentionnés dans la bibliographie. Nous tenons d'ailleurs à remercier ici les membres du groupe *comp.lang.java.help* pour l'aide qu'il nous ont offerte.

Pour nous, donc, un nouveau langage. Mais les nouveautés ne s'arrêtaient pas là : le type même de programmation auquel il fallait faire face était inédit pour nous, puisqu'il s'agissait de programmation réseau, en particulier d'architecture Client/Serveur. Par bonheur, Java dispose de ressources nombreuses et efficaces dans ce domaine, qui nous ont permis de porter notre attention davantage sur l'apprentissage et le développement que sur les difficultés à résoudre, et ainsi de parvenir sans trop de mal à un résultat concret.

A ce niveau, le seul regret que nous ayons à exprimer concerne le fait que le mémoire n'a pas été réalisé par deux étudiants, comme c'était originellement prévu, mais par un seul. Travailler à deux eût permis de diviser la charge de recherche et de rédaction, en laissant de cette façon plus de latitude pour le développement de l'application qui s'en fût trouvée, à n'en pas douter, plus étoffée.

Nous ne prétendons pas avoir atteint à la maîtrise complète des sujets que nous avons abordés, mais nous affirmons avoir trouvé dans leur étude un intérêt supérieur à l'obligation académique. Nous espérons que cet intérêt a transparu au fil des pages qui composent ce mémoire, et nous souhaitons l'avoir fait partager au lecteur, que nous remercions ici pour le temps et l'attention qu'il a portés au parcours de ce travail.

Abréviations

- *ATM* : Asynchronous Transfer Mode
- *CASE* : Computer Aided Software Engineering
- *CERN* : Conseil Européen pour la Recherche Nucléaire
- *DNS* : Domain Name Service
- *DoD* : Department of Defense
- *DTD* : Document Type Definition (Définition de Type de Document)
- *DTMF* : Dual Tone Multi-Frequency
- *EDP* : Electronic Data Processing
- *ETSI* : European Telecommunications Standardization Institute
- *FCC* : Federal Communications Commission
- *FTP* : File Transfer Protocol
- *HTML* : Hyper Text Markup Language
- *HTTP* : Hyper Text Transfer Protocol
- *IHL* : Internet Header Length (dans un datagramme IP)
- *IN* : Intelligent Network
- *IP* : Intelligent Peripheral
- *IP* : Internet Protocol
- *ISDN* : Integrated Services Digital Netwotk
- *ISO* : International Organization for Standardization
- *IT* : Information Technology
- *ITU* : International Telecommunication Union
- *NSF* : National Science Foundation
- *PDU* : Protocol Data Unit
- *PIN* : Personal Identification Number
- *PSPDN* : Packet Switched Public Data Network
- *PSTN* : Public Switched Telephone Network
- *RFC* : Request for Comments
- *RTC* : Réseau Téléphonique Commuté
- *SCE* : Service Creation Environment
- *SCP* : Service Control Point
- *SDL* : System Description Language
- *SIB* : Service-Independent Building Block
- *SMP* : Service Management Point
- *SQL* : Structured Query Language
- *SSP* : Service Switching Point
- *TCP* : Transmission Control Protocol
- *UDP* : User Datagram Protocol
- *URL* : Uniform Resource Locator
- *VPN* : Virtual Private Network
- *WWW* : World-Wide *Web*

Bibliographie

1. Ouvrages

- Alcatel Telecommunications Review – Intelligent Networks : opening the door to new services, 1st Quarter 1996.
- December, J., Morrison, M. *et al.*, *JAVA - Secrets d'experts*, Simon & Schuster Macmillan, Paris, 1996.
- Lemay, L., Perkins, Ch. L., *Apprenez Java 1.1 en 21 jours*, Simon & Schuster Macmillan, coll. Le Programmeur, Paris, 1997.
- Lips, B., *Internet en Belgique*, Best Of Éditions, Bruxelles, 1996.
- Nachtergaele, V. et van Bastelaer, Ph., *Cours de compléments de téléinformatique*, Deuxièmes Licence et Maîtrise en Informatique, FUNDP, septembre 1997.
- Stanek, R. S., *HTML, JAVA, CGI, VRML, SGML - Secrets d'experts*, Simon & Schuster Macmillan, Paris, 1996.
- Tanenbaum, A., *Réseaux - Architectures, protocoles, applications*, InterEditions, Paris, 1990.
- Van den Bossche, M., *Still points of a turning world*, conférence donnée à l'Institut d'informatique, Namur, mai 1998.
- Van Herwijnen, E., *SGML Pratique*, International Thomson Publishing France, Paris, 1995.

2. Ressources *on-line*

- comp.lang.java.help
- java.sun.com
- search.zdnet.com
- www.bellcore.com
- www.cio.com
- www.gamelan.com
- www.iscp.com
- www.javasoft.com
- www.javaworld.com
- www.pcwebopaedia.com
- www.tra.com
- www.webforum.com

Annexes

1. Exemples

1.1. Exemples de services IN

1.1.1. Advanced freephone, Universal access number, Kiosk, Automatic call distribution

Cette série de services offre des facilités en matière de taxation et de routage flexibles.

La taxation flexible signifie que l'appel peut être à l'entière charge du souscripteur du service appelé, ou que son prix peut être réparti, dans une certaine proportion, entre la partie appelante et la partie appelée.

Le routage flexible signifie que l'appel n'est pas routé seulement sur la base du numéro formé, mais aussi en fonction de l'origine de l'appel, de l'heure, du jour, de la date, *etc.*

Des combinaisons de taxation et de routage flexibles sont possibles.

1.1.2. Credit card billing, Alternate billing, Prepaid card call

Ce type de service permet à l'utilisateur de faire des appels téléphoniques qui sont facturés de différentes manières. La facture peut être envoyée à une partie tierce, par exemple l'entreprise à laquelle appartient l'utilisateur dans le cas d'appels de nature professionnelle ; la facture peut être partagée en fonction de différents paramètres, par exemple à des fins promotionnelles ; la facture peut être établie *via* un compte ouvert chez le fournisseur de service ou *via* une carte de crédit classique ; elle peut être prise en charge *via* une carte prépayée, par exemple lors de voyages ou à des fins promotionnelles ; *etc.*

La carte en question peut être une « vraie » carte à utiliser avec un terminal muni d'un lecteur de cartes, ou elle peut être une carte « virtuelle » où seul le numéro qu'elle porte doit être composé par l'utilisateur. Chacun de ces deux types de carte peut, pour des questions de sécurité, être utilisé en conjonction avec un PIN (*Personal Identification Number*) qui, une fois composé par l'utilisateur, permettra à l'appel d'avoir lieu.

Ces services comprennent des vérifications et de statistiques sur l'utilisation de la carte de crédit (liste noire, ligne de crédit totale, crédit par appel, consommation, *etc.*).

1.1.3. Personal number, Universal personal telecommunications

Ce type de service vise à augmenter la mobilité du souscripteur de service dans la mesure où il peut indiquer (enregistrer) la destination (c'est-à-dire le numéro de téléphone) à laquelle il peut être joint.

Un appel à destination de son numéro personnel sera routé automatiquement à l'endroit de son dernier enregistrement.

1.1.4. Virtual Private network

Ce service est particulièrement utile aux sociétés désirant les facilités d'un réseau privé, avec un investissement nul ou minimal en PABX (*Private Automatic Branch Exchange*) et en lignes louées.

Le réseau public et les services IN leur permettent de définir un plan de numérotation privé, une topologie et une configuration de réseau privé en termes de nombre d'extensions pour chaque site.

1.1.5. Vote and opinion poll, Mass calling

Ce type de service est surtout utilisé lors d'activités telles que les jeux radiophoniques ou télévisés pour lesquels les gens peuvent voter ou exprimer une opinion, et qui provoquent généralement un trafic téléphonique énorme.

La surcharge peut être contrôlée en implémentant ce service comme un service IN, particulièrement là où les centraux locaux agissent comme des SSP (*Service Switching Points*). Dans ce cas, le SCP peut donner l'ordre aux centraux locaux de limiter le nombre d'appels en fonction de la capacité de la destination à les gérer.

1.2. Création d'un service

Le service considéré est un service de *card calling* avec code d'identification. Son déroulement est le suivant : l'utilisateur du service compose un préfixe déterminé et, averti par le réseau, compose le numéro de

la carte et le numéro de l'appel qu'il désire effectuer. L'appel est ensuite routé vers sa destination à travers le réseau si la carte existe et est valide.

A sa création, le service est décrit comme une séquence de SIB (*Service Independent building Blocks*). Cette séquence se compose de SIB qui assurent les fonctions suivantes :

1. envoi d'un avis vocal et collecte des chiffres entrés par l'utilisateur (par exemple, l'utilisateur entend « Composez votre code confidentiel. », puis le fait) ;
2. vérification de l'existence de la carte ;
3. vérification du code confidentiel ;
4. connexion de l'appel ;
5. supervision de l'appel (ce SIB surveille l'appel, prêt à détecter un événement tel une des parties qui raccroche).

Les SIB assurant ces fonctions sont agencés comme suit. Le service débute par un SIB de départ qui est suivi de deux SIB qui demandent à l'utilisateur d'indiquer son numéro de carte et son code confidentiel PIN. Ensuite un SIB vérifie que la carte existe et un autre vérifie le code PIN. Si toutes ces vérifications sont satisfaites, un SIB demande à l'utilisateur de composer le numéro de l'appel qu'il veut effectuer. Un SIB établit alors l'appel, tandis qu'un autre SIB le supervise. Ce dernier SIB effectue les tâches nécessaires si l'une ou l'autre des parties raccroche (s'il s'agit de la partie appelante, le service s'achève ; si c'est la partie appelée qui raccroche, l'utilisateur peut souhaiter faire un autre appel, dans ce cas, il n'est pas obligé de raccrocher).

2. Compléments sur les protocoles

2.1. Le protocole IP

2.1.1. Adressage IP

Toute machine hôte est identifiée au sein du réseau Internet par une adresse unique formée d'un entier codé sur 32 bits et appelée *adresse IP*.[62] Cette adresse résulte de la concaténation de deux identifiants :

1. *netid* : identifiant du sous-réseau auquel appartient la machine ;
2. *hostid* : identifiant de cette machine au sein du sous-réseau.

Les 32 bits de l'adresse IP sont souvent décomposés en quatre groupes de 8 bits représentés chacun par l'entier qui leur correspond (a.b.c.d). En outre, l'ensemble des adresses IP est réparti en 3 classes qui définissent chacune un format précis pour les identifiants *netid* et *hostid* :

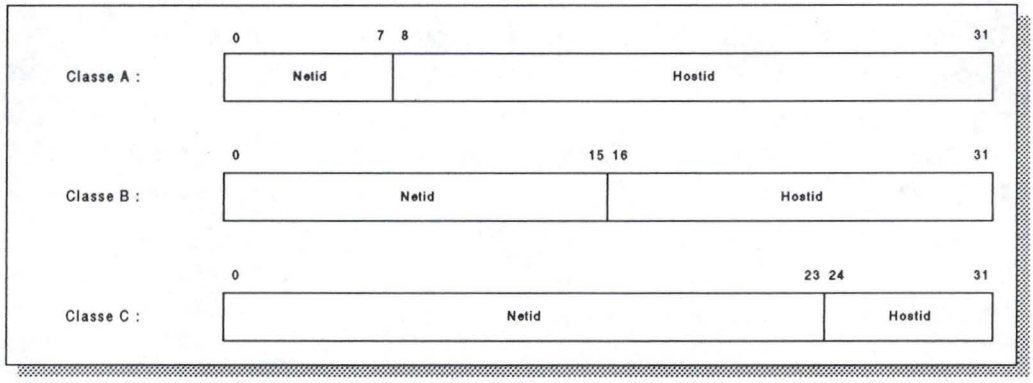


Figure 12 : Classes d'adresses IP

- La **classe A** caractérise des sous-réseaux pouvant être formés d'au plus 2^{24} (soit 16 777 216) machines : l'identifiant *netid* se compose des bits 0 à 7 et l'identifiant *hostid* des bits 8 à 31. Les adresses IP de classe A ont toutes leur premier bit (le bit 0) à 0 : il peut donc y avoir un maximum de 2^7 (soit 128) sous-réseaux distincts de classe A. Au sein de cette classe, un sous-réseau est identifié par a ($0 \leq a < 128$), et chaque machine d'un sous-réseau est identifiée par b.c.d.
- La **classe B** caractérise des sous-réseaux pouvant être formés d'au plus 2^{16} (soit 65 536) machines : l'identifiant *netid* se compose des bits 0 à 15 et l'identifiant *hostid* des bits 16 à 31. Les adresses IP de classe B ont toutes leur premier bit (le bit 0) à 1 et leur second bit (le bit 1) à

[62] En plus de son adresse IP, une machine dispose souvent d'un nom plus facilement mémorisable. Un protocole appelé DNS (*Domain Name Service*) permet d'établir la correspondance entre le nom d'une machine et son adresse IP.

0 : il peut donc y avoir un maximum de 2^{14} (soit 16 384) sous-réseaux distincts de classe B. Au sein de cette classe, un sous-réseau est identifié par a.b ($128 \leq a < 192$), et chaque machine d'un sous-réseau est identifiée par c.d.

- La **classe C** caractérise des sous-réseaux pouvant être formés d'au plus 2^8 (soit 256) machines : l'identifiant *netid* se compose des bits 0 à 24 et l'identifiant *hostid* des bits 25 à 31. Les adresses IP de classe C ont toutes leur premier bit (le bit 0) à 1 et leur second bit (le bit 1) à 1 : il peut donc y avoir un maximum de 2^{21} (soit 2 097 152) sous-réseaux distincts de classe C. Au sein de cette classe, un sous-réseau est identifié par a.b.c ($192 \leq a < 256$), et chaque machine d'un sous-réseau est identifiée par d.

2.1.2. Primitives de service IP

Le protocole IP assurant un service *connectionless*, il n'y a pas de primitive d'établissement ni de clôture de connexion, mais une primitive SEND de type *request*, permettant l'envoi de données sur le réseau par la machine émettrice, et une primitive DELIVER de type *indication* utilisée par la machine réceptrice lors de la réception d'unités de données.

Les paramètres communs à ces deux primitives sont les suivants : l'adresse de la source ainsi que celle de la destination, le protocole utilisé dans la partie données du datagramme IP (souvent TCP ou UDP), des indicateurs de type de service requis pour l'unité de données (par exemple un traitement préférentiel du datagramme, le niveau de fiabilité attendu en termes de perte de données et de taux d'erreur, la minimisation du délai de transit et la maximisation de la vitesse de transmission), la longueur du champ de données, des options de qualité de service (comme la spécification d'une route à suivre, l'enregistrement de la route suivie par le datagramme à travers le réseau, un label de sécurité, *etc.*) et enfin les données utilisateur à transmettre.

La primitive SEND comporte en outre les champs particuliers suivants : un identifiant optionnel du datagramme permettant d'identifier les entités concernées au niveau supérieur, un champ indiquant si le datagramme peut ou non être fragmenté par la couche IP et un indicateur du « temps » maximum que peut mettre un datagramme pour arriver à sa destination (le nombre de passerelles par lesquelles il peut être traité avant d'être éliminé)

L'ensemble des paramètres de ces primitives de service est présenté dans le tableau suivant :

Primitives de service	Paramètres	Type
SEND Demande de transfert de données	Source address Destination address Protocol Type of service indicators Identifier Don't fragment indicator Time to live Data length Option Data	Request
DELIVER Livraison des données reçues	Source address Destination address Protocol Type of service indicators Data length Option Data	Indication

Figure 13 : Primitives de service IP

2.1.3. Format du datagramme IP

Selon le protocole IP, le transfert, par l'entité IP de la machine source, de données en provenance de l'entité Transport d'une machine hôte se fait vers l'entité IP réceptrice de la machine de destination sous la forme d'unités de protocole de la couche IP, appelées *datagrammes*. La figure suivante indique le format des datagrammes IP :

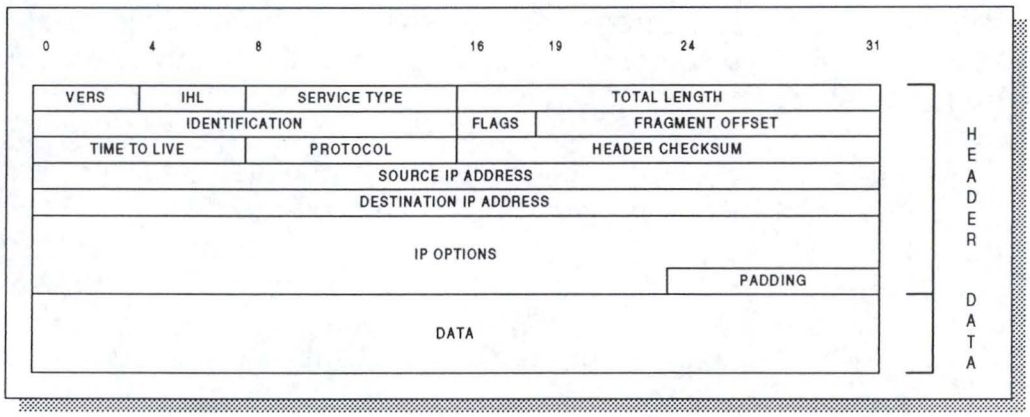


Figure 14 : Format du datagramme IP

Les différents champs qui composent le datagramme du protocole IP sont les suivants :

- VERS (4 bits) : donne la version du protocole IP utilisé (actuellement la version d'IP est 4) ;

- IHL (*Internet Header Length*, 4 bits) : indique la longueur de l'en-tête du datagramme (multiple de 32 bits) ;
- SERVICE TYPE (8 bits) : indique le type de service demandé aux algorithmes de routage (temps, fiabilité, débit) ;
- TOTAL LENGTH (16 bits) : donne la longueur totale du datagramme (octets de 8 bits) ;
- IDENTIFICATION (16 bits) : présente le numéro d'identification du datagramme ;
- FLAGS (3 bits) : se réfère au mécanisme de fragmentation (le premier bit, *More Flag*, indique si le datagramme est le résultat d'une fragmentation ; le second bit, *Don't Fragment*, interdit la fragmentation s'il est à 1 ; le troisième n'est pas employé actuellement) ;
- FRAGMENT OFFSET (13 bits) : indique, lors de la fragmentation d'un datagramme, la position, en terme d'octets de 8 bits, du fragment à l'intérieur du datagramme originel ;
- TIME TO LIVE : indique le temps maximum, en secondes, durant lequel le datagramme peut transiter sur le réseau. Chaque fois que le datagramme atteint une passerelle, ce champ est décrémenté du temps en secondes passé par le datagramme dans cette passerelle. Quand la valeur de ce champ atteint 0, le datagramme est éliminé ;
- PROTOCOL (8 bits) : indique le protocole utilisé dans la partie données du datagramme (généralement TCP ou UDP) ;
- HEADER CHECKSUM (16 bits) : vérifie que l'en-tête du datagramme n'a pas été corrompu. Lorsqu'une erreur est détectée par une entité IP sur un en-tête de datagramme, celui-ci est éliminé ;
- SOURCE IP ADDRESS (32 bits) : donne l'adresse IP de la machine hôte sur laquelle réside l'entité IP émettrice ;
- DESTINATION IP ADDRESS (32 bits) : donne l'adresse IP de la machine hôte de destination ;
- IP OPTIONS : permet de spécifier les options sélectionnées par l'émetteur (champ facultatif et de longueur variable) ;
- PADDING : contient une suite de bits permettant de compléter le champ IP OPTIONS de façon à ce que la longueur de celui-ci soit multiple de 32 bits.

2.2. Le protocole TCP

2.2.1. Primitives de service TCP

Le protocole TCP, assurant un service en mode connecté, avec contrôle d'erreur sur les données et contrôle de flux sur les segments transmis, dispose de nombreuses primitives.

La première série de primitives concerne tout ce qui a trait à la connexion TCP : ouvertures passive et active (UNSPECIFIED_PASSIVE_OPEN,

FULL_PASSIVE_OPEN, ACTIVE_OPEN, ACTIVE_OPEN_WITH_DATA, OPEN_ID, OPEN_SUCCESS, OPEN_FAILURE), libération (CLOSE, CLOSING, TERMINATE, ABORT) et état (STATUS, STATUS_RESPONSE). La seconde série de primitives concerne le transfert des données : envoi (SEND), livraison (DELIVER), gestion de la fenêtre (ALLOCATE) et erreurs (ERROR).

Le tableau suivant reprend l'ensemble de ces primitives, avec une brève description de leur rôle et la liste de leur paramètres[63] :

a) Primitives liées à la connexion TCP

1. Ouvertures passive et active

Primitives de service	Paramètres	Type
UNSPECIFIED_PASSIVE_OPEN Demande d'ouverture passive de connexion non spécifiée, avec un niveau de sécurité et une précédence donnés	Source port [timeout, timeout-action, precedence, security range]	Request
FULL_PASSIVE_OPEN Demande d'ouverture passive de connexion, avec indication du processus actif attendu avec un niveau de sécurité et une précédence donnés	Source port Destination port Destination address [timeout, timeout-action, precedence, security range]	Request
ACTIVE_OPEN Demande d'ouverture active de connexion, avec un niveau de sécurité et une précédence donnés pour une destination donnée	Source port Destination port Destination address [timeout, timeout-action, precedence, security range]	Request
ACTIVE_OPEN_WITH_DATA Demande d'ouverture active de connexion, avec un niveau de sécurité et une précédence donnés pour une destination donnée, avec transmission simultanée de données	Source port Destination port Destination address Data Data length Push flag Urgent flag [timeout, timeout-action, precedence, security range]	Request

[63] Les paramètres entre crochets sont optionnels.

OPEN_ID Signalement de la prise en compte de la demande d'ouverture (passive ou active) de connexion et communication de l'identifiant de cette connexion « pendante »	Local connection name Source port [destination port, destination address][64]	Local response
OPEN_SUCCESS Signalement de la réussite de l'ouverture de connexion	Local connection name	Confirm
OPEN_FAILURE Signalement de l'échec de l'ouverture de connexion	Local connection name	Confirm

Figure 15 : Primitives liées à la connexion TCP – ouvertures passive et active

2. Libération

Primitives de service	Paramètres	Type
CLOSE Demande de libération ordonnée de la connexion	Local connection name	Request
CLOSING Signalement au processus utilisateur de TCP que le processus utilisateur éloigné a effectué une demande de libération de connexion et que donc toutes les données envoyées par ce dernier ont été reçues	Local connection name	Indication
TERMINATE Signalement de la bonne terminaison de la phase de libération de la connexion TCP de nom donné	Local connection name Reason code	Confirm
ABORT Demande de libération brutale de la connexion, sans attendre que les données encore en attente soient transmises	Local connection name	Request

Figure 16 : Primitives liées à la connexion TCP – libération

[64] Optionnels dans le cas de la demande d'ouverture passive de connexion.

3. Etat

Primitives de service	Paramètres	Type
STATUS Communication d'une description de l'état de la connexion (adresse, port du destinataire et de la source de la connexion, nombre de buffers en attente de confirmation, sécurité, timeout, ...)	Local connection name	Request
STATUS_RESPONSE Communication d'une description d'une erreur survenue, qu'il s'agisse d'une erreur interne ou d'une erreur de service	Local connection name Source port Source address Destination port Destination address Connection state Receive window Send windows Waiting ack Waiting receipt Urgent Precedence Security Timeout	Local response

Figure 17 : Primitives liées à la connexion TCP – état

b) Primitives liées au transfert de données

1. Envoi et livraison

Primitives de service	Paramètres	Type
SEND Demande de transfert de données sur la connexion identifiée par son nom	Local connection name Data Data length Push flag Urgent flag [timeout, timeout action]	Request
DELIVER Livraison des données reçues sur la connexion identifiée par son nom	Local connection name Data Data length Urgent flag	Indication

Figure 18 : Primitives liées au transfert de données - envoi et livraison

2. Gestion de la fenêtre et erreurs

Primitives de service	Paramètres	Type
ALLOCATE Demande de modification de la taille de la fenêtre de réception des données	Local connection name Data length	Request
ERROR Signalement au processus utilisateur de TCP d'une situation d'erreur	Local connection name Reason code	Indication

Figure 19 : Primitives liées au transfert de données - gestion de la fenêtre et erreurs

c) Paramètres des primitives de service

- *Destination address* : adresse IP de la machine hôte destinatrice.
- *Source & destination ports* : paramètres identifiant respectivement le port identifiant le processus d'application émetteur sur la machine hôte émettrice et le port identifiant le processus d'application récepteur sur la machine hôte réceptrice.
- *Local connection name* : nom local identifiant la connexion, fourni par l'entité TCP au processus d'application lors de la phase d'établissement de la connexion TCP avec un processus d'application sur une machine distante. Cet identifiant est ensuite utilisé durant toute la durée de vie de la connexion par le processus d'application, en lieu et place de l'adresse complète du destinataire.
- *Timeout & timeout action* : paramètres précisant la durée du délai (*timeout*) nécessaire à la confirmation de la livraison des données et après lequel une action (*timeout action*) doit être entreprise (coupure brutale de la connexion ou retransmission des données).
- *Precedence* : paramètre permettant au processus d'application de négocier avec la couche TCP et le processus homologue un certain niveau de priorité (dans une gamme allant de 1 à 7) que les processus d'application attendent sur la connexion TCP.
- *Security range* : paramètre permettant au processus d'application de négocier avec la couche TCP et le processus homologue un certain niveau de sécurité qu'il désire obtenir sur la connexion TCP (code de contrôle de transmission, traitement d'exception, *etc.*).
- *Push flag & urgent flag* : paramètres permettant à un processus d'application d'exiger le transfert immédiat des données par l'entité TCP émettrice (*push flag*) ou d'exiger la mise en œuvre du mécanisme de transfert des données urgentes (*urgent flag*).
- *Data & data length* : bloc de données transmis d'un processus d'application vers l'autre (*data*) et sa longueur (*data length*).

2.2.2. Format du segment TCP

Selon le protocole TCP, le transfert, par l'entité TCP de la machine source, de données en provenance d'un processus d'application d'une machine hôte se fait vers l'entité TCP réceptrice de la machine de destination sous la forme d'unités de protocole de la couche Transport, appelées *segments*. La figure suivante indique le format des segments TCP :

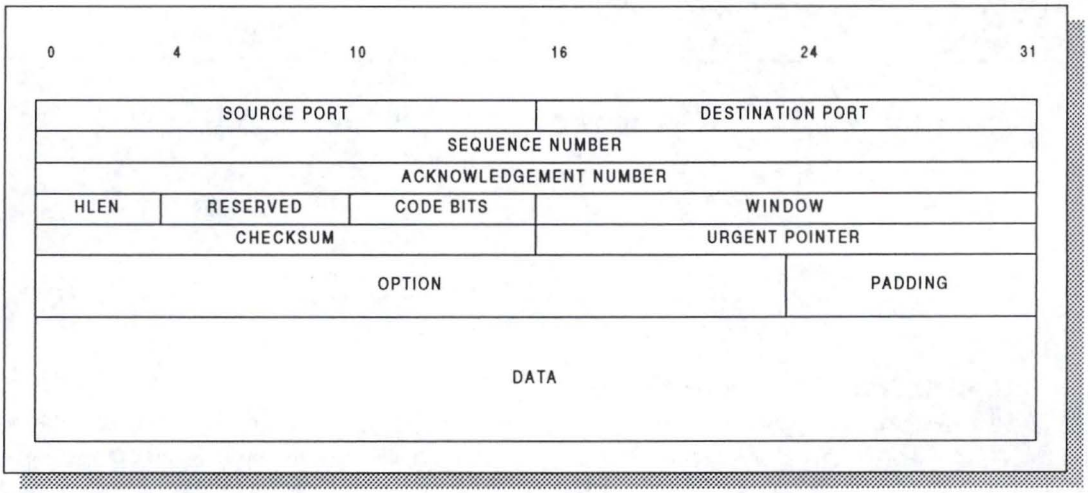


Figure 20 : Format du segment TCP

Les différents champs qui composent le segment TCP sont les suivants :

- SOURCE PORT (16 bits) : port utilisé sur la machine source ;
- DESTINATION PORT (16 bits) : port à utiliser sur la machine de destination ;
- SEQUENCE NUMBER (32 bits) : position du premier octet de la partie des données du segment dans le flot d'octets soumis pour transmission par la machine hôte source ;
- ACKNOWLEDGEMENT NUMBER (32 bits) : acquittement (par la machine émettrice du segment le contenant) de tous les octets de numéro strictement inférieur à la valeur de ce champ, relatifs au flot de données reçu par cette dernière. Ce nombre représente donc le numéro du prochain octet que la machine réceptrice s'attend à recevoir ;
- HLEN (4 bits) : longueur de l'en-tête (variable à cause du champ OPTION) mesurée en multiple de 32 bits ;
- RESERVED (6 bits) : champ réservé pour une utilisation future ;
- CODE BITS (6 bits) : champ déterminant le but et le contenu du segment[65] ;

[65] La signification des 6 bits de ce champ (de gauche à droite) est reprise dans la table ci-dessous :

- WINDOW (16 bits) : champ pouvant servir, à chaque échange d'un segment TCP, à changer la taille de la fenêtre utilisée pour le contrôle de flux ;
- CHECKSUM (16 bits) : champ servant à vérifier l'intégrité de l'en-tête et des données ;
- URGENT POINTER (16 bits) : longueur des données urgentes envoyées ;
- OPTION : champ de longueur variable servant à négocier certaines options de la connexion (taille maximum d'un segment, sécurité, *etc.*) ;
- PADDING : suite de bits de longueur variable servant à ce que la longueur de l'en-tête soit multiple de 32.

Il est à noter que tout segment TCP contient l'adresse complète du processus d'application destinataire (adresse IP et port, soit un *socket*), ainsi que le port identifiant le processus d'application source. Le nom local de connexion fourni par l'entité TCP au processus d'application lors de la phase d'établissement de la connexion TCP est utilisé par celui-ci lors du transfert des données, et c'est à l'entité TCP d'établir, par des tables, la correspondance entre ce nom local de connexion et l'adresse du destinataire telle qu'elle doit apparaître dans le segment TCP.

2.3. Le protocole HTTP

2.3.1. Primitives de service HTTP

Il faut distinguer le service offert par le client WWW (le *browser*) à l'utilisateur, le service offert par le client HTTP au client WWW et le service offert par le serveur HTTP.

Au niveau du protocole HTTP, les standards ne mentionnent pas explicitement les primitives de service abstraites. Nous suivrons donc le formalisme (nom des primitives et choix des paramètres) adopté par Ph. van Bastelaer dans son cours de Matières Approfondies de Télécommunications.[66]

Bit	Signification dans le cas où le bit est à 1
URG	Le champ URGENT POINTER est valide
ACK	Le champ ACKNOWLEDGEMENT est valide
PSH	Ce segment demande un PUSH
RST	Ce segment coupe la connexion
SYN	Ce segment synchronise les numéros de séquence durant la phase d'ouverture de la connexion
FIN	La source a terminé l'envoi de ses données

[66] van Bastelaer, Ph., *Chapitre WWW. Niveau 7 : Le World-Wide Web, HTTP et HTML*, avril 97, in Nachtergaele, V. et van Bastelaer, Ph., *Cours de compléments de téléinformatique*, *op. cit.*

a) Primitives côté client

1. Service offert par le client WWW

Primitives de service	Paramètres	Req	Conf	Ind	Resp
OPEN Demande de recherche d'un document	URL du document Document obtenu Statut du transfert	X	 X X		
FORM (cas interactif) Demande d'envoi d'un formulaire	URL du script Formulaire Réponse du script Statut	X X	 X X		

Figure 21 : Primitives côté client – service offert par le client WWW

2. Service offert par le client HTTP

Primitives de service	Paramètres	Req	Conf	Ind	Resp
GET Demande de recherche d'un document	URL du document ou partie de document Document obtenu Statut du transfert	X	 X X		
POST (cas interactif) Demande d'exécution d'un script	URL du script Informations issues du formulaire Informations complémentaires	X X X			

Figure 22 : Primitives côté client – service offert par le client HTTP

b) Primitives côté serveur

1. Service offert par le serveur HTTP

Primitives de service	Paramètres	Req	Conf	Ind	Resp
GET Demande de fourniture d'un document	Path du document Informations complémentaires Document obtenu Statut			X X	 X X X
POST (cas interactif) Demande d'exécution d'un script	Path du script Informations issues du formulaire Informations complémentaires			X X X	

Figure 23 : Primitives côté serveur – service offert par le serveur HTTP

2. Service offert par le serveur WWW

Primitives de service	Paramètres	Req	Conf	Ind	Resp
FORM Demande d'exécution	Informations issues du formulaire Réponse du script Statut de l'exécution			X	 X X

Figure 24 : Primitives côté serveur – service offert par le serveur WWW

2.3.2. Format du PDU HTTP

Rappelons que HTTP a, entre autres, pour but de permettre l'échange d'une **requête** et d'une **réponse** entre un **client** et un **serveur**. A cette fin, le protocole HTTP dispose de deux formats de PDU (*Protocol Data Unit*), l'un associé aux requêtes, l'autre associé aux réponses.

Le format d'un PDU du protocole HTTP relatif aux requêtes est le suivant :

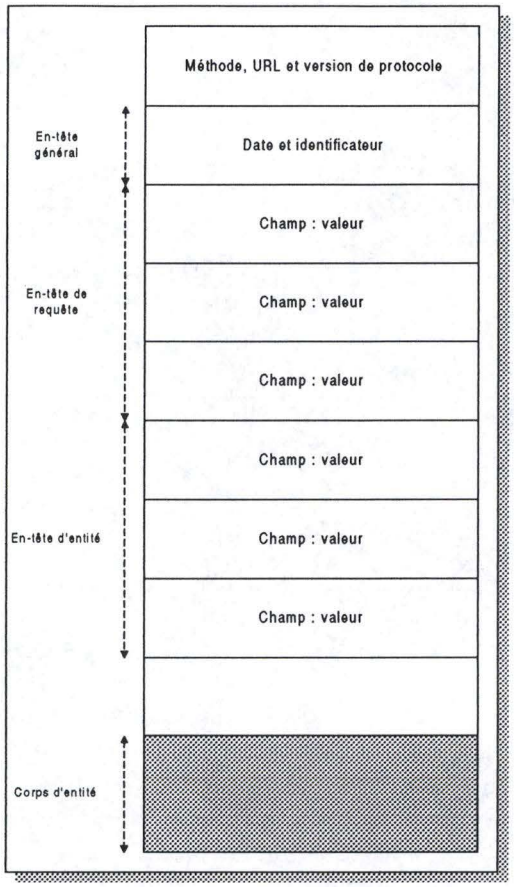


Figure 25 : Format d'un PDU HTTP de requête

Le format d'un PDU du protocole HTTP relatif aux réponses est le suivant :

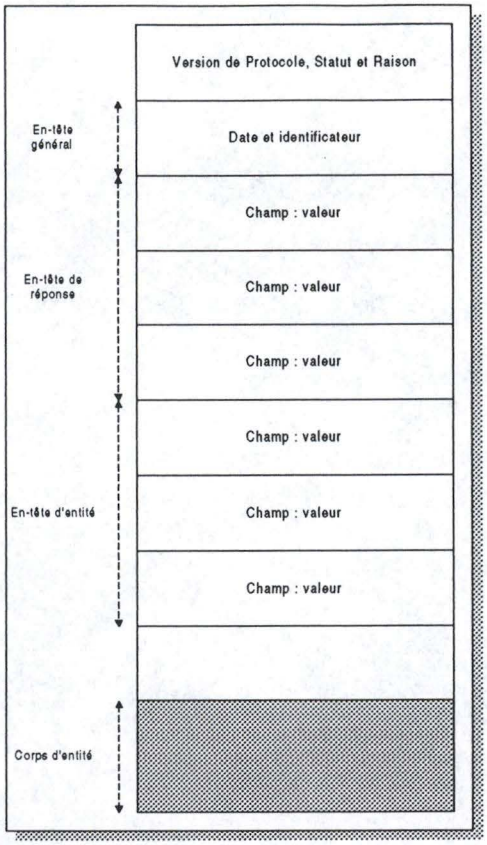


Figure 26 : Format d'un PDU HTTP de réponse

Il est à remarquer que les champs, en HTTP, sont de longueur variable : une ligne ne correspond donc pas nécessairement à un octet.

Dans le cas d'un PDU de requête, la première ligne contient l'identifiant de la méthode (par exemple POST, GET), l'URL (partiel si c'est suffisant) de la ressource désirée (document ou script) et la version du protocole utilisé (par exemple HTTP/1.0). Elle est éventuellement suivie d'un **en-tête général** composé d'un champ contenant la date et d'un champ d'identification de la requête. Les lignes suivantes constituent l'**en-tête de requête** et contiennent un nombre quelconque de couples <champ : valeur>, optionnels et en ordre indifférent (par exemple ACCEPT, FROM, *etc.*). Le champ de données ou « entité » contient d'abord un **en-tête d'entité** optionnel contenant, par exemple, la longueur et le type d'entité, puis un **corps d'entité** composé d'un nombre arbitraire de données.

Dans le cas d'un PDU de réponse, la première ligne contient la version du protocole utilisé, un statut codé sur un entier de trois chiffres et suivi d'une courte phrase explicitant ce code (par exemple 400 Bad Request). Elle est éventuellement suivie d'un **en-tête général** composé d'un champ contenant la date et d'un champ d'identification de la requête. Les lignes suivantes constituent l'**en-tête de réponse** et contiennent un nombre quelconque de couples <champ : valeur> en ordre quelconque (par exemple Retry After, Server, *etc.*).

3. Code de l'implémentation

3.1. Le générateur d'événements

3.1.1. Première approche : la classe Generateur

```
/*
###############
# Generateur #
###############
*/


// Imports

import java.io.*;
import java.lang.Math;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.text.SimpleDateFormat;


/*
=====================
= Classe Generateur =
=====================
*/


public class Generateur {


  /*
  ------------------
  - Méthode main() -
  ------------------
  */

  public static void main (String args[]) throws IOException {

    int nb_alarmes = 0; // nombre d'alarmes désiré
    DataOutput alarmes = new DataOutputStream(new
      FileOutputStream("alarmes.txt")); // Fichier dans lequel seront
      écrites les alarmes
    Locale loc = new Locale("FRENCH", "FRANCE"); // Mise en forme de
      la date
    SimpleDateFormat formatDate = new SimpleDateFormat("'le'
      dd/MM/yyyy 'a' HH:mm:ss", loc); // Mise en forme de la date


    // Vérification de la conformité de l'argument

    String erreur = "Usage : java Generateur <nombre d'alarmes
      desire> (1-1000)";

    if (args.length!=1) {
      System.out.println(erreur);
```

```
      System.exit(1);
    }
    try {nb_alarmes = Integer.parseInt(args[0]);}
    catch (NumberFormatException e) {
      System.out.println(erreur);
      System.exit(1);
    }
    if ((nb_alarmes<1) | (nb_alarmes>1000)) {
      System.out.println(erreur);
      System.exit(1);
    }


    //Génération des alarmes

    System.out.println("");
    System.out.println("");
    System.out.println("ALARMES GENEREES PAR LE RESEAU (fichier
      alarmes.txt) : ");
    System.out.println("");

    Random r     = new Random();  // Générateur de nombres aléatoires
    String line1 ="";
    String date1 ="";
    int    rd1   = 0;             // niveau d'alarme
    int    rd2   = 0;             // type d'alarme

    for (int i = 1; i<=nb_alarmes; i++) {
      // Insérer ici, éventuellement, un retardateur aléatoire
      Date date = new Date();
      date1     = formatDate.format(date);

      rd1       = (Math.abs(r.nextInt())%15+1); // 15 niveaux
        d'alarme possibles
      rd2       = (Math.abs(r.nextInt())%10+1); // 10 types d'alarme
        possibles

      line1     = "Alarme numero " + Integer.toString(i) + " de
        niveau " + Integer.toString(rd1);
      line1     = line1 + " et de type " + Integer.toString(rd2);
      line1     = line1 + " echue " + date1;

      alarmes.writeChars(line1+"\n"); // Ecriture des alarmes dans le
        fichier
      System.out.println(line1);  // Affichage des alarmes à l'écran

    } // Fin de for

    System.out.println("");
    System.out.println("_____");
    System.out.println("");
    Date dateCourante = new Date();
    String dateString = formatDate.format(dateCourante);
    System.out.println(dateString);

  } // Fin de main()

} // Fin de class Generateur
```

3.1.2. Seconde approche : la classe Alarme

```
/*

##########
# Alarme #
##########

*/


// Imports

import java.io.*;
import java.lang.Math;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.text.SimpleDateFormat;


/*

==================
= Classe Alarme =
==================

*/


public class Alarme {

  // Variables d'instance

  int numero = 0;
  int niveau = 0;
  int type   = 0;
  Date date  = null;


  // Initialisation du format de date et du générateur de nombres
     aléatoires

  static Locale loc = new Locale("FRENCH", "FRANCE");
  static SimpleDateFormat formatDate = new SimpleDateFormat("'le'
    dd/MM/yyyy 'a' HH:mm:ss", loc);
  Random r = new Random();

  /*
  ---------------------------
  - Méthode générerAlarme() -
  ---------------------------
  */

  void générerAlarme() {

    Date dateCourante = new Date();

    date   = dateCourante;
    niveau = (Math.abs(r.nextInt())%15+1); // 15 niveaux d'alarme
      possibles
    type   = (Math.abs(r.nextInt())%10+1); // 10 types d'alarme
      possibles
```

```
} // fin générerAlarme()


/*
----------------------------
- Méthode afficherAlarme() -
----------------------------

    La méthode afficherAlarme() prend pour arguments un entier qui
    sera le numéro de l'alarme (correspondant à la valeur de i dans
    la boucle for de la méthode main()) et le fichier alarmes.txt
    dans lequel seront écrites les alarmes.
*/

void afficherAlarme(int x, DataOutput dataOut) throws IOException {

  String line1 ="";
  String date1 ="";

  line1 = "Alarme numero " + Integer.toString(x) + " de niveau " +
    Integer.toString(niveau);
  line1 = line1 + " et de type " + Integer.toString(type);
  line1 = line1 + " echue " + formatDate.format(date);

  dataOut.writeChars(line1+"\n"); // Ecriture des alarmes dans le
    fichier
  System.out.println(line1);      // Affichage des alarmes à
    l'écran

} // fin afficherAlarme();


/*
------------------
- Méthode main() -
------------------

    La méthode main() prend pour argument le nombre d'alarmes que
    l'utilisateur souhaite faire générer. Elle vérifie d'abord la
    conformité de l'argument, puis utilise celui-ci comme compteur
    pour la boucle principale (for) qui emploie les méthodes
    générerAlarme() et afficherAlarme() définies ci-dessus.
*/

public static void main(String[] args) throws IOException {

  Alarme alarme = new Alarme();

  // Fichier dans lequel seront écrites les alarmes
  DataOutput alarmes = new DataOutputStream(new
    FileOutputStream("alarmes.txt"));

  // Vérification de la conformité de l'argument

  String erreur = "Usage : java Generateur <nombre d'alarmes
    desire> (1-1000)";
  int nb_alarmes = 0;

  if (args.length!=1) {
    System.out.println(erreur);
    System.exit(1);
  }
  try {nb_alarmes = Integer.parseInt(args[0]);}
```

```
    catch (NumberFormatException e) {
      System.out.println(erreur);
      System.exit(1);
    }
    if ((nb_alarmes<1) | (nb_alarmes>1000)) {
      System.out.println(erreur);
      System.exit(1);
    }

    System.out.println("");
    System.out.println("");
    System.out.println("ALARMES GENEREES PAR LE RESEAU : ");
    System.out.println("");

    for (int i = 1; i<=nb_alarmes; i++) {

      alarme.générerAlarme();
      alarme.afficherAlarme(i, alarmes);

    } // fin de boucle for

    System.out.println("");
    System.out.println("_____________________");
    System.out.println("");
    Date dateCourante = new Date();
    String dateString = formatDate.format(dateCourante);
    System.out.println(dateString);

  } // fin de main(args)

} // fin de class Alarme
```

3.2. Le générateur d'événements muni d'un filtre

```
/*

##########
# Alarme #
##########

*/


// Imports

import java.io.*;
import java.lang.Math;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.text.SimpleDateFormat;


/*

=================
= Classe Alarme =
=================

*/
```

```
public class Alarme {


  // Variables d'instance

  int     numero      = 0;
  int     niveau      = 0;
  int     type        = 0;
  Date    date        = null;
  boolean filtrée     = false;
  boolean prioritaire = false;


  // Initialisation du format de date et du générateur de nombres
     aléatoires

  static Locale loc = new Locale("FRENCH", "FRANCE");
  static SimpleDateFormat formatDate = new SimpleDateFormat("'le'
    dd/MM/yyyy 'a' HH:mm:ss", loc);
  Random r = new Random();

  /*
  ---------------------------
  - Méthode générerAlarme() -
  ---------------------------
  */

  void générerAlarme() {

    Date dateCourante = new Date();

    date   = dateCourante;
    niveau = (Math.abs(r.nextInt())%15+1); // 15 niveaux d'alarme
      possibles
    type   = (Math.abs(r.nextInt())%10+1); // 10 types d'alarme
      possibles

  } // fin générerAlarme()

  /*
  ---------------------------
  - Méthode filtrerAlarme() -
  ---------------------------

      La méthode filtrerAlarme() décide si les alarmes sont filtrées
      et prioritaires sur base des arguments indiqués par
      l'utilisateur et fournis à la méthode filtrerAlarme() par la
      méthode main().
  */

  void filtrerAlarme(int niv, int typ, int niv_p, int typ_p) {
    if ((niveau<=niv) && (type<=typ))
      filtrée = true;
    else filtrée = false;
    if ((niveau<=niv_p) | (type<=typ_p))
      prioritaire = true;
    else prioritaire = false;
  } // fin filtrerAlarme(niv, typ)

  /*
  ---------------------------
  - Méthode afficherAlarme() -
  ---------------------------
```

```
    L'ensemble des alarmes sont écrites dans le fichier alarmes.txt
    et affichées à l'écran, les alarmes filtrées sont écrites dans
    le fichier resultats.txt, marquées puis affichées à l'écran.
    Les alarmes prioritaires sont marquées puis affichées à 'écran.
*/

void afficherAlarme(int x, DataOutput dataOutAl, DataOutput
  dataOutRes) throws IOException {

  String line1 ="";

  line1 = "Alarme numero " + Integer.toString(x) + " de niveau " +
    Integer.toString(niveau);
  line1 = line1 + " et de type " + Integer.toString(type);
  line1 = line1 + " echue " + formatDate.format(date);

  dataOutAl.writeChars(line1+"\n");     // Ecriture de l'alarme dans
    le fichier alarmes.txt

  if (filtrée) {
    dataOutRes.writeChars(line1+"\n"); // Ecriture de l'alarme dans
      le fichier resultats.txt
    line1 = line1 + "(f)"; // Marquage de l'alarme filtrée
  }

  if (prioritaire) {
    line1 = line1 + "(p)"; // Marquage de l'alarme filtrée
  }

  System.out.println(line1); // Affichage des alarmes à l'écran

} // fin afficherAlarme(x, dataOutAl, dataOutRes);


/*
------------------
- Méthode main() -
------------------

    La méthode main() définit un nouvel objet alarme et les flux de
    fichiers alarmes.txt et resultats.txt. Elle vérifie ensuite les
    arguments fournis par l'utilisateur en ligne de commande et les
    range dans les variables correspondantes. Enfin, la boucle for,
    qui s'exécute le nombre de fois demandé par l'utilisateur,
    utilise les trois méthodes définies précédemment,
    générerAlarme(), filtrerAlarme() et afficherAlarme(), pour
    l'exécution du logiciel.
*/

public static void main(String[] args) throws IOException {

  Alarme alarme = new Alarme();

  // Fichier dans lequel seront écrites les alarmes
  DataOutput alarmes = new DataOutputStream(new
    FileOutputStream("alarmes.txt"));

  // Fichier dans lequel seront écrites les alarmes filtrées
  DataOutput resultats = new DataOutputStream(new
    FileOutputStream("resultats.txt"));

  int nb_al = 0; // nombre d'alarmes désiré
  int n     = 0; // niveau d'alarme minimal à partir duquel on
```

```
  désire être prévenu
int t     = 0; // type d'alarme minimal à partir duquel on désiré
  être prévenu
int n_p   = 0; // niveau d'alarme minimal prioritaire à partir
  duquel on désire être prévenu
int t_p   = 0; // type d'alarme minimal prioritaire à partir
  duquel on désiré être prévenu

// Vérification de la conformité des arguments

String erreur = "Usage : java Generateur <nombre d'alarmes
  desire> (1-1000), <niveau d'alarme> (1-5), <type d'alarme> (1-
  4), <niveau d'alarme prioritaire> (1-5), <type d'alarme
  prioritaire> (1-4)";

if (args.length!=5) {
  System.out.println(erreur);
  System.exit(1);
} // fin de if

try {nb_al = Integer.parseInt(args[0]);
     n     = Integer.parseInt(args[1]);
     t     = Integer.parseInt(args[2]);
     n_p   = Integer.parseInt(args[3]);
     t_p   = Integer.parseInt(args[4]);
} // fin de try
catch (NumberFormatException e) {
  System.out.println(erreur);
  System.exit(1);
} // fin de catch

if ((nb_al<1) | (nb_al>1000) |
    (n    <1) | (n    >5)    |
    (t    <1) | (t    >4))   {
  System.out.println(erreur);
  System.exit(1);
} // fin de if


// Génération des alarmes

System.out.println("");
System.out.println("");
System.out.println("ALARMES GENEREES PAR LE RESEAU (fichier
  alarmes.txt) : ");
System.out.println("(les alarmes filtrees sont marquees de (f),
  les alarmes prioritaires sont marquees de (p) et sont ecrites
  dans le fichier resultats.txt)");
System.out.println("");

for (int i = 1; i<=nb_al; i++) {            // boucle principale

  alarme.générerAlarme();
  alarme.filtrerAlarme(n, t, n_p, t_p);
  alarme.afficherAlarme(i, alarmes, resultats);

} // fin de boucle for

System.out.println("");
System.out.println("_____________________");
System.out.println("");
Date dateCourante = new Date();
String dateString = formatDate.format(dateCourante);
System.out.println(dateString);
```

```
  } // fin de main(args)

} // fin de class Alarme
```

3.3. L'architecture Client/Serveur

3.3.1. Le serveur

```
/*

############
# Serveur #
############

*/


// Imports

import java.io.*;
import java.net.*;
import java.util.Random;


/*

==================
= Classe Serveur =
==================

*/


public class Serveur extends Thread {


  // Déclaration et initialisation des variables

  private static final int PORTNUM  = 5000; // numéro de port
  private ServerSocket serverSocket = null; // socket serveur


  /*
  --------------------------
  - Constructeur Serveur() -
  --------------------------

      Le constructeur appelle le constructeur de la classe parent
      (thread) et essaie de créer un socket serveur. L'utilisateur
      est averti du succès ou de l'échec de l'opération.
  */

  public Serveur() {

    super("Serveur");

    try {
      serverSocket = new ServerSocket(PORTNUM);
      System.out.println("Le serveur tourne... (" +
```

```
        serverSocket.getInetAddress() + " : " +
        serverSocket.getLocalPort() + ")");
    } // fin de try

    catch (IOException e) {
      System.err.println("Exception: impossible de creer un socket");
      System.exit(1);
    } // fin de catch

} // fin du constructeur Serveur()


/*
----------------------
- Méthode main(args) -
----------------------

    La méthode main(), sans argument, crée une nouvelle instance de
    la classe Serveur et la fait démarrer.
*/

public static void main(String[] args) {

  Serveur serveur = new Serveur();
  serveur.start();

} // fin de main()


/*
-----------------
- Méthode run() -
-----------------

    La méthode run(), sans argument, est, outre la méthode main(),
    la méthode principale de la classe Serveur. Elle teste d'abord
    si le socket serveur a été créé. La méthode accept de la classe
    ServerSocket retourne, lorsqu'un client entre en contact avec
    le serveur, un objet Socket qui représente la connexion.
*/

public void run() {

  Socket  clientSocket = null;

  // Boucle principale

  while (true) {

    // Attente d'un client

    if (serverSocket == null)
      return;
    try {
      clientSocket = serverSocket.accept();
    }
    catch (IOException e) {
      System.err.println("Exception: impossible de se connecter au
        socket client");
      System.exit(1);
    }
    System.err.println("Connexion recue de " +
      clientSocket.getInetAddress() + " : " +
      clientSocket.getPort());
```

```
        // Exécution des fonctionnalités

        try {

        // Définition des flux

        BufferedReader is = new BufferedReader(new
          InputStreamReader(clientSocket.getInputStream()));
        PrintWriter os = new PrintWriter   (new
          BufferedOutputStream(clientSocket.getOutputStream()), false);
        String  inLine  = "";
        String  outLine = "";
        boolean done    = false;

        os.println("Hello !");
        os.flush();

        while (true) {
          if (!(inLine = (is.readLine()).trim()).equals("Salut")) {
            os.println(inLine); // Fonction perroquet côté client
            os.flush();
            System.out.println("Client : " + inLine); // Fonction
              perroquet côté serveur
            inLine = "";
          } // fin de if
          else {
           os.close();                // Nettoyage
           is.close();
             clientSocket.close();
             System.exit(0);
          } // fin de else

        } // fin de while

        } // fin du try

        catch (IOException e) {
          System.err.println("Exception :" + e);
        e.printStackTrace();
        } // fin de catch

      } // fin de while(true)

    } // fin de run()

} // fin de class Serveur
```

3.3.2. Le client

```
/*

##########
# Client #
##########

*/

// Imports

import java.io.*;
```

```
import java.net.*;

/*

=================
= Classe Client =
=================

*/

public class Client {

  // Déclaration et initialisation des variables

  private static final int PORTNUM = 5000;  // numéro de port

  /*
  ----------------------
  - Méthode main(args) -
  ----------------------

     La méthode main() déclare et initialise un socket qui
     représentera la connexion avec le serveur, et deux flux, l'un
     pour les entrées, l'autre pour les sorties. Une chaîne est
     utilisée pour l'adresse du serveur. Elle vérifie ensuite la
     conformité de l'argument (l'adresse du serveur), puis elle
     initialise le socket à l'adresse donnée et les flux de et vers
     ce socket.
  */

  public static void main(String[] args) {

    Socket          socket  = null;
    BufferedReader  in      = null;
    PrintWriter     out     = null;
    String          adresse = "";


    // Vérification de la ligne de commande

    if (args.length != 1) {
      System.out.println("Usage: java Client <adresse>");
      return;
    }
    else
      adresse = args[0];


    // Initialisation du socket et des flux

    try {
      socket = new Socket(adresse, PORTNUM);
      in = new BufferedReader(new
        InputStreamReader(socket.getInputStream()));
      out = new PrintWriter(socket.getOutputStream());
    } // fin du try

    catch (IOException e) {
      System.err.println("Exception: impossible de creer le flux de
        socket");
```

```
      System.exit(1);
    } // fin du catch


    // Exécution des fonctionnalités

    try {

      // Traitement des échanges
      StringBuffer str = new StringBuffer(128);
      String     inStr = "";
      int            c = 0;

      while((inStr = in.readLine()) != null) {
        System.out.println("Serveur : " + inStr);
        if (inStr.equals("Salut"))
        break;
        while ((c=System.in.read()) != '\n')
          str.append((char)c); // Lecture des caractères entrés par
            l'utilisateur (Client), stockés dans str
        System.out.println("Client : " + str); // Affichage des
          caractères entrés par l'utilisateur (Client)
        out.println((str.toString()).trim()); // Envoi des caractères
          au serveur
        out.flush();
        str.setLength(0);
      } // fin du while

      // Nettoyage

      out.close();
      in.close();
      socket.close();

    } // fin du try

    catch (IOException e) {
      System.err.println("Exception: Erreur E/S en tentant de
        communiquer avec le Serveur");
    } // fin de catch

  } // fin de main()

} // fin de class Client
```

3.4. Le Serveur et le Client définitifs

3.4.1. Le serveur

```
/*

############
# Serveur #
############

*/
```

```
// Imports

import java.io.*;
import java.net.*;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.lang.Math;
import java.lang.System;
import java.text.SimpleDateFormat;


/*

==================
= Classe Serveur =
==================

*/


public class Serveur extends Thread {


  // Déclaration et initialisation des variables

  private static final int  PORTNUM = 5000; // numéro de port
  private ServerSocket serverSocket = null; // socket serveur


  // Initialisation du format de date et du générateur de nombres
     aléatoires

  static Locale loc = new Locale("FRENCH", "FRANCE");
  static SimpleDateFormat formatDate = new SimpleDateFormat("'le'
    dd/MM/yyyy 'a' HH:mm:ss", loc);
  Random r = new Random();

  /*
  --------------------------
  - Constructeur Serveur() -
  --------------------------

      Le constructeur appelle le constructeur de la classe parent
      (thread) et essaie de créer un socket serveur. L'utilisateur
      est averti du succès ou de l'échec de l'opération.
  */

  public Serveur() {

    super("Serveur");

    try {
      serverSocket = new ServerSocket(PORTNUM);
      System.out.println("Le serveur tourne... (" +
        serverSocket.getInetAddress() + " : " +
        serverSocket.getLocalPort() + ")");
    } // fin de try

    catch (IOException e) {
      System.err.println("Exception: impossible de creer un socket");
      System.exit(1);
    } // fin de catch
```

```
} // fin du constructeur Serveur()


/*
----------------------
- Méthode main(args) -
----------------------

    La méthode main() crée une nouvelle instance de la classe
    Serveur et la fait démarrer.
*/

public static void main(String[] args) {

  Serveur serveur = new Serveur();
  serveur.start();

} // fin de main()


/*
-----------------
- Méthode run() -
-----------------

    La méthode run(), sans argument, est, outre la méthode main(),
    la méthode principale de la classe Serveur. Elle teste d'abord
    si le socket serveur == null. La méthode accept de la classe
    ServerSocket retourne un objet Socket qui représente la
    connexion. Les autres fonctionnalités sont commentées à
    l'endroit où elles sont codées.
*/

public void run() {

  Socket  clientSocket = null;

  // Boucle principale

  while (true) {

    // Attente d'un client

    if (serverSocket == null)
      return;
    try {
      clientSocket = serverSocket.accept();
    }
    catch (IOException e) {
      System.err.println("Exception: impossible de se connecter au
        socket client");
      System.exit(1);
    }
    System.err.println("Connexion recue de " +
      clientSocket.getInetAddress() + " : " +
      clientSocket.getPort());

    // Exécution des fonctionnalités

    try {

    // Définition des flux

    BufferedReader  is = new BufferedReader(new
```

```
  InputStreamReader(clientSocket.getInputStream()));
PrintWriter     os = new PrintWriter   (new
  BufferedOutputStream(clientSocket.getOutputStream()), true);

String inLine   = ""; // String entrante
String outLine  = ""; // String sortante

// String d'erreur de nom
String err_nom  = "Vous n'avez pas acces au monitoring";

// String d'erreur de mot de passe
String err_pwd  = "Votre mot de passe n'est pas correct";

String nom      = ""; // Nom de l'utilisateur
String passe    = ""; // Mot de passe de l'utilisateur

int ident = 0;  // Identifiant du réseau intelligent
int type  = 0;  // Type d'alarme minimal à partir duquel
  l'utilisateur désire être informé (à échéance)
int niveau = 0;  // Niveau d'alarme minimal à partir duquel
  l'utilisateur désire être informé (à échéance)
int type_p = 0;  // Type d'alarme minimal prioritaire à partir
  duquel l'utilisateur désire être informé (hors échéance)
int niveau_p = 0;  // Niveau d'alarme minimal prioritaire à
  partir duquel l'utilisateur désire être informé (hors
  échéance)
int fréq     = 0;  // Fréquence du monitoring
int durée    = 0;  // Durée du monitoring
String dém   = ""; // Démarrage du monitoring
Int cycles   = 0;  // Nombre de cycles de monitoring
  (durée/fréquence)
int étapes   = 7;  // Nombre d'étapes de l'acquisition des
  paramètres que l'utilisateur doit fournir
int[] paramètres = new int[étapes]; // Tableau de rangement des
  paramètres indiqués par l'utilisateur



// Identification de l'utilisateur


// Bienvenue et nom

os.println("Bonjour ! Entrez votre nom :");
nom = (is.readLine()).trim();
if (!(nom.equals("Benoit"))) { // Vérif. du nom de l'utilisat.
  os.println("Erreur : " + err_nom);
  System.exit(1);
}
System.out.println("Client : nom = " + nom);


// Mot de passe

os.println("Entrez votre mot de passe :");
passe = (is.readLine()).trim();
if (!(passe.equals("passe"))) { // Vérif. du pwd de l'utilis.
  os.println("Erreur : " + err_pwd);
  System.exit(1);
}
System.out.println("Client : passe = " + passe);


// Acquisition des paramètres (un tableau contient l'ensemble
```

```
        des paramètres)

      for (int i = 0 ; i < étapes ; i++) {
        dialogue(is, os, i, paramètres);
      } // Fin de for (int i = 0, i < étapes, i++)


      // Assignation des paramètres à leurs variables respectives

      ident    = paramètres[0];
      type     = paramètres[1];
      niveau   = paramètres[2];
      type_p   = paramètres[3];
      niveau_p = paramètres[4];
      fréq     = paramètres[5];
      durée    = paramètres[6];


      // Démarrage du monitoring

      os.println("Demarrer le monitoring ? (o/n) :");
      dém = is.readLine();
      if (dém.equals("o")) {
        monitoring(os, type, niveau, type_p, niveau_p, fréq, durée);
        if (is.readLine().equals("exit")) {
          os.println("Fin du monitoring");
          os.close();
          is.close();
            clientSocket.close();
          System.exit(0);
        }
      } // Fin de if (dém.equals("o"))
      else {
        os.println("Fin du monitoring.");
        os.close();
        is.close();
          clientSocket.close();
        System.exit(0);
      } // Fin de else

      } // fin du try

      catch (IOException e) {
        System.err.println("Exception :" + e);
      e.printStackTrace();
      } // fin de catch

    } // fin de while(true)

  } // fin de run()


  /*
  ------------------------------------------------------------------------
  - Méthode dialogue(BufferedReader in, PrintWriter out, int étape, -
  - int[] params)                                                        -
  ------------------------------------------------------------------------

      La méthode dialogue() gère le premier échange de données entre
      le client et le serveur. Le client fournit au serveur les
      paramètres nécessaires à l'exécution du monitoring. A chaque
      entrée de l'utilisateur, la méthode vérification() est appelée
      afin de s'assurer de la conformité des arguments entrés par
      l'utilisateur.
```

```
*/

public void dialogue(BufferedReader in, PrintWriter out, int étape,
  int[] params) throws IOException {

  String[] phrasesClient = {
    "Entrez l'identifiant de l'IN :",
    "Entrez le type d'alarme minimal (1-15) :",
    "Entrez le niveau d'alarme minimal (1-10) :",
    "Entrez le type d'alarme minimal prioritaire (< type d'alarme
      minimal) :",
    "Entrez le niveau d'alarme minimal prioritaire (< niveau
      d'alarme minimal) :",
    "Entrez la frequence du monitoring (en secondes) :",
    "Entrez la duree du monitoring (en secondes, > frequence) :"
  };

  String[] phrasesServeur = {     "ident",
                                  "type",
                                  "niveau",
                                  "type_p",
                                  "niveau_p",
                                  "freq",
                                  "duree"
  };

  String[] erreurs = {
    "Le monitoring n'est accessible que pour le reseau intelligent
      no 1.",
    "Le type d'alarme minimal doit etre compris entre 1 et 15.",
    "Le niveau d'alarme minimal doit etre compris entre 1 et 10.",
    "Le type d'alarme minimal prioritaire doit etre compris entre 1
      et 15 et inferieur ou egal au type d'alarme minimal.",
    "Le niveau d'alarme minimal prioritaire doit etre compris entre
      1 et 10 et inferieur ou egal au niveau d'alarme minimal.",
    "La frequence doit etre un entier",
    "La duree doit etre superieure a la frequence."
  };

  int inter = 0; // Stockage intermédiaire de l'entrée de
    l'utilisateur à des fins de vérification


  // Déroulement du dialogue

  out.println(phrasesClient[étape]);

  try {inter = Integer.parseInt((in.readLine()).trim());
  } // Fin de try

  catch (NumberFormatException e) {
    out.println("Vous devez entrer un nombre.");
    out.println("Fin du monitoring.");
    System.exit(1); // Placer ici un mécanisme de retour au
      dialogue (reprise après erreur)
  } // Fin de catch

  // Vérification de la conformité des arguments fournis par
     l'utilisateur

  if (vérification(étape, inter, params)) params[étape] = inter;
  else {
    out.println("Erreur : " + erreurs[étape]);
```

```
        out.println("Fin du monitoring.");
        System.exit(1); // Placer ici un mécanisme de retour au
          dialogue (reprise après erreur)
      } // Fin de else

      System.out.println("Client : " + phrasesServeur[étape] + " = " +
        params[étape]);

    } // Fin de dialogue (BufferedReader in, PrintWriter out, int
étape, int[] params)


    /*
    ------------------------------------------------------------------
    - Méthode vérification (int étape_, int inter_, int[] params_) -
    ------------------------------------------------------------------

        La méthode vérification(), sur appel de la méthode dialogue(),
        s'assure que les données fournies par l'utilisateur pour
        l'exécution du monitoring correspondent à ce que l'on attend.
    */

    public boolean vérification (int étape_, int inter_, int[] params_)
{

      switch (étape_) {
        case 0 : return (inter_ == 1);
        case 1 : return (inter_ > 0 && inter_ < 16);
        case 2 : return (inter_ > 0 && inter_ < 11);
        case 3 : return (inter_ > 0 && inter_ < 16 && inter_ <=
          params_[1]);
        case 4 : return (inter_ > 0 && inter_ < 11 && inter_ <=
          params_[2]);
        case 5 : return true;
        case 6 : return (inter_ > params_[5]);
        default: return false;
      } // Fin de switch (étape_)

    } // Fin de vérification (int étape_, int inter_, int[] params_)


    /*
    ------------------------------------------------------------------
    - Méthode monitoring(PrintWriter out, int t, int n, int t_p, int -
    - n_p, int f, int d)                                              -
    ------------------------------------------------------------------

        La méthode monitoring() fonctionne de façon analogue à la
        méthode main() de la classe Alarme. Une instance de cette
        classe est créée et des appels sont faits aux méthodes
        générerAlarme(), filtrerAlarme() et afficherAlarme() de cette
        classe. Ces appels sont effectués au sein d'une boucle for qui
        compte le nombre de cycles de monitoring à faire, qui contient
        elle-même une boucle while qui assure la génération, le
        filtrage et l'affichage côté serveur des alarmes (rem. : le
        filtrage envoie aussi côté client les alarmes prioritaires). La
        boucle while se termine à l'échéance d'un cycle de monitoring.
        A ce moment, les alarmes filtrées sont envoyées au client par
        la méthode envoyerAlarmes().
    */

    public void monitoring(     PrintWriter out,
                                int t,
                                int n,
```

```
                                int t_p,
                                int n_p,
                                int f,
                                int d       ) throws IOException {


// Variables se rapportant à l'exécution du monitoring

int cycles = Math.round(d/f); // Nombre de cycles de monitoring à
  effectuer (durée/fréquence)
long heureDébut = System.currentTimeMillis(); // Heure de début
  du monitoring
long heureCourante = 0; // Heure courante
long échéance      = 0; // Prochaine échéance du monitoring
int  compteur      = 1; // Compteur d'alarmes
String fichierAl   = ""; // Nom du fichier qui contiendra les
  alarmes
String fichierRes  = ""; // Nom du fichier qui contiendra les
  alarmes filtrées

Alarme alarme = new Alarme(); // Instance de la classe Alarme


out.println("Monitoring..."); // Signal de début pour le client

for (int i = 1 ; i <= cycles ; i++) {

  fichierAl  = "al"  + i + ".txt";
  fichierRes = "res" + i + ".txt";

  // Fichier dans lequel seront écrites les alarmes
  DataOutput alarmes = new DataOutputStream(new
    FileOutputStream(fichierAl));

  // Fichier dans lequel seront écrites les alarmes filtrées
  DataOutput resultats = new DataOutputStream(new
    FileOutputStream(fichierRes));

  int[]  alNum    = new int  [1000];
  int[]  alType   = new int  [1000];
  int[]  alNiveau = new int  [1000];
  Date[] alDate   = new Date [1000];
  int      index = 0;

  while ((heureCourante = System.currentTimeMillis()) < (échéance
    = heureDébut + (i*f*1000))) {

  // Générer une alarme
  alarme.générerAlarme();
  alarme.numero = compteur;

  // Filtrer une alarme (la stocker dans un tableau d'alarmes si
     elle est filtrée
  alarme.filtrerAlarme(t, n, t_p, n_p);

  // Afficher une alarme côté serveur (optionnel), l'envoyer côté
     client si elle est prioritaire, stockage dans les fichiers
     alarmes.txt et resultats.txt
  alarme.afficherAlarme(compteur, alarmes, resultats);
  if (alarme.prioritaire) {
    out.println("Alarme prioritaire (num " + compteur + ", type "
      + alarme.type + ", niveau " + alarme.niveau + ", le " +
      formatDate.format(alarme.date));
  }
```

```
      // Incrémentation du compteur d'alarmes
      compteur++;

      // Préparer les alarmes pour la méthode envoyerAlarmes()
      if (alarme.filtrée) {
        alNum[index]    = alarme.numero;
        alType[index]   = alarme.type;
        alNiveau[index] = alarme.niveau;
        alDate[index]   = alarme.date;
        index++;
      }


      } // Fin de while ((heureCourante = System.currentTimeMillis())
          < (heureDébut + (i*f*1000)))


      // Envoyer Alarmes
      envoyerAlarmes(out, i, alNum, alType, alNiveau, alDate);

    } // Fin de for (int i = 1, i <= cycles, i++)

    out.println("Fin du monitoring."); // Signal de fin pour le
      client

  } // Fin de monitoring(PrintWriter out, int t, int n, int t_p, int
n_p, int f, int d)



  /*
  ------------------------------------------------------------------
  - Méthode envoyerAlarmes(PrintWriter pw, int numCycle, int[] -
  - tabNum, int[] tabType, int[] tabNiveau, Date[] tabDate)    -
  ------------------------------------------------------------------

  */

  void envoyerAlarmes(PrintWriter pw, int numCycle, int[] tabNum,
    int[] tabType, int[] tabNiveau, Date[] tabDate) {

    int x = 0;

    System.out.println("");
    System.out.println("Envoi des alarmes filtrees (cycle " +
      numCycle + ").");
    System.out.println("");

    pw.println("");
    pw.println("Alarmes filtrees...");
    pw.println("Cycle " + numCycle);
    pw.println("");

    while (tabNum[x] != 0) {
      pw.println("Alarme filtree num " + tabNum[x] + ", type " +
        tabType[x] + ", niv " + tabNiveau[x] + ", date : " +
        formatDate.format(tabDate[x]));
      x++;
    }

    pw.println("");
    pw.println("Fin de l'envoi des alarmes filtrees (cycle " +
      numCycle + ").");
```

```
    pw.println("");

  } // Fin de envoyerAlarmes(PrintWriter pw, int numCycle, int[]
      tabNum, int[] tabType, int[] tabNiveau, Date[] tabDate)

} // fin de class Serveur
```

3.4.2. Le client

```
/*

##########
# Client #
##########

*/

// Imports

import java.io.*;
import java.net.*;


/*

=================
= Classe Client =
=================

*/


public class Client {

  // Déclaration et initialisation des variables

  private static final int PORTNUM = 5000;  // numéro de port

  /*
  ----------------------
  - Méthode main(args) -
  ----------------------

      La méthode main() déclare et initialise un socket qui
      représentera la connexion avec le sereveur, et deux flux, l'un
      pour les entrées, l'autre pour les sorties. Une chaîne est
      utilisée pour l'adresse du serveur. Elle vérifie ensuite la
      conformité de l'argument (l'adresse du serveur), puis elle
      initialise le socket à l'adresse donnée et les flux de et vers
      ce socket.
  */

  public static void main(String[] args) throws IOException {

    Socket          socket    = null;
    BufferedReader  in        = null;
    PrintWriter     out       = null;
    String          adresse   = "";

    // Fichier dans lequel seront écrites les alarmes filtrées
    DataOutput      resultats = new DataOutputStream(new
```

```
  FileOutputStream("resultats.txt"));

// Vérification de la ligne de commande

if (args.length != 1) {
  System.out.println("Usage: java Client <adresse>");
  return;
}
else
  adresse = args[0];


// Initialisation du socket et des flux

try {
  socket = new Socket(adresse, PORTNUM);
  in = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
  out = new PrintWriter(socket.getOutputStream(), true);
} // fin du try

catch (IOException e) {
  System.err.println("Exception: impossible de creer le flux de
    socket");
  System.exit(1);
} // fin du catch


// Exécution des fonctionnalités

try {

  // Traitement des échanges
  StringBuffer str = new StringBuffer(128);
  String     inStr = "";
  int            c = 0;

  /*
  Le dialogue entre le client et le serveur se fait d'abord sur
  la base d'un échange phrase après phrase (identification de
  l'utilisateur et acquisition des paramètres). Ensuite, quand le
  monitoring commence, le serveur a besoin de pouvoir communiquer
  un ensemble de données au client (les alarmes), sans que celui-
  ci doive répondre. La commutation entre les deux types de
  dialogue se fait sur la base d'indicateurs fournis par le
  serveur.
  */


  while((inStr = in.readLine()) != null) {

  System.out.println(inStr);

  if (inStr.startsWith("Erreur : ")) {
      out.close();
      in.close();
      socket.close();
    System.exit(0);
  }

  if (inStr.equals("Monitoring...")) {
    try {
      while (!(inStr = in.readLine()).equals("Fin du
        monitoring.")) {
```

```
            System.out.println(inStr);
            if (inStr.startsWith("Alarme filtree")) {
              resultats.writeChars(inStr + '\n');
            } // Fin de if (inStr.startsWith("Alarme filtree")
          } // Fin de while (!(inStr = in.readLine()).equals("Fin du
              monitoring."))
        } // Fin de try
        catch (IOException e) {
          System.exit(1);
        }
      } // Fin de if (inStr.equals("Monitoring..."))

      while ((c=System.in.read()) != '\n') {
        str.append((char)c); // Lecture des caractères entrés par
        l'utilisateur (Client), stockés dans str
      } // Fin de while ((c=System.in.read()) != '\n')

      System.out.println("Client : " + str); // Affichage des
        caractères entrés par l'utilisateur (Client)
      out.println((str.toString()).trim());  // Envoi des caractères
        au serveur
      str.setLength(0);

      if (inStr.equals("Fin du monitoring.")) {
          out.close();
          in.close();
          socket.close();
        System.exit(0);
      }

      } // fin de while((inStr = in.readLine()) != null)


      // Nettoyage

      out.close();
      in.close();
      socket.close();


    } // fin du try


    catch (IOException e) {
      System.err.println("Exception: Erreur E/S en tentant de
        communiquer avec le Serveur");
      e.printStackTrace();

    } // fin du catch

  } // fin de main()

} // fin de class Client
```

3.4.3. La classe Alarme

```
/*

##########
# Alarme #
##########

*/


// Imports

import java.io.*;
import java.lang.Math;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.text.SimpleDateFormat;


/*

=================
= Classe Alarme =
=================

*/


public class Alarme {


  // Variables d'instance

  int     numero      = 0;
  int     type        = 0;
  int     niveau      = 0;
  Date    date        = null;
  boolean filtrée     = false;
  boolean prioritaire = false;


  // Initialisation du format de date et du générateur de nombres
     aléatoires

  static Locale loc = new Locale("FRENCH", "FRANCE");
  static SimpleDateFormat formatDate = new SimpleDateFormat("'le'
    dd/MM/yyyy 'a' HH:mm:ss", loc);
  Random r = new Random();


  /*
  ---------------------------
  - Méthode générerAlarme() -
  ---------------------------

  */

  void générerAlarme() {

    Date dateCourante = new Date();
```

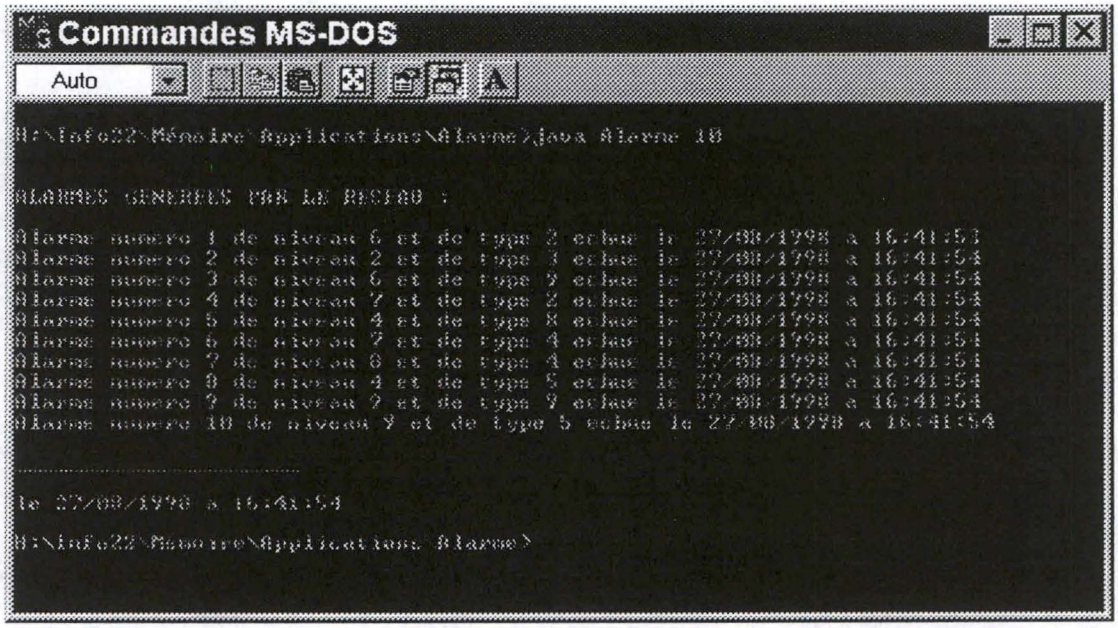
```
    date    = dateCourante;
    type    = (Math.abs(r.nextInt())%15+1); // 15 types d'alarme
      possibles
    niveau = (Math.abs(r.nextInt())%10+1); // 10 niveaux d'alarme
      possibles

  } // fin générerAlarme()


  /*
  ---------------------------
  - Méthode filtrerAlarme() -
  ---------------------------

      Les alarmes sont filtrées selon un niveau et un type spécifiés
      par l'utilisateur
  */

  void filtrerAlarme(int typ, int niv, int typ_p, int niv_p) {

    if ((type<=typ) && (niveau<=niv))
      filtrée = true;
    else filtrée = false;

    if ((type<=typ_p) | (niveau<=niv_p))
      prioritaire = true;
    else prioritaire = false;

  } // fin filtrerAlarme(niv, typ)


  /*
  ----------------------------
  - Méthode afficherAlarme() -
  ----------------------------

      L'ensemble des alarmes sont écrites dans le fichier alarmes.txt
      et affichées à l'écran, les alarmes filtrées sont écrites dans
      le fichier resultats.txt, marquées puis affichées à l'écran.
  */

  void afficherAlarme(int x, DataOutput dataOutAl, DataOutput
    dataOutRes) throws IOException {

    String line1 ="";

    line1 = "Alarme numero " + Integer.toString(x) +
          " de type " + Integer.toString(type) +
          " et de niveau " + Integer.toString(niveau) +
          " echue " + formatDate.format(date);;

    dataOutAl.writeChars(line1+'\n');     // Ecriture de l'alarme dans
      le fichier alarmes.txt

    if (filtrée) {
      line1 = line1 + " (f)"; // Marquage de l'alarme filtrée
    }

    if (prioritaire) {
      line1 = line1 + " (p)"; // Marquage de l'alarme prioritaire
    }

    if (filtrée | prioritaire) {
      dataOutRes.writeChars(line1+'\n'); // Ecriture de l'alarme
```

```
        filtrée ou prioritaire dans le fichier resultats.txt
    }

    System.out.println(line1); // Affichage des alarmes à l'écran

} // fin afficherAlarme(x, dataOutAl, dataOutRes);


/*
------------------
- Méthode main() -
------------------

    La méthode main() définit un nouvel objet alarme et les flux de
    fichiers alarmes.txt et resultats.txt. Elle vérifie ensuite les
    arguments fournis par l'utilisateur en ligne de commande et les
    range dans les variables correspondantes. Enfin, la boucle for,
    qui s'exécute le nombre de fois demandé par l'utilisateur,
    utilise les trois méthodes définies précédemment,
    générerAlarme(), filtrerAlarme() et afficherAlarme(), pour
    l'exécution dulogiciel.
*/

public static void main(String[] args) throws IOException {

    Alarme alarme = new Alarme();

    // Fichier dans lequel seront écrites les alarmes
    DataOutput alarmes   = new DataOutputStream(new
      FileOutputStream("alarmes.txt"));
    // Fichier dans lequel seront écrites les alarmes filtrées
    DataOutput resultats = new DataOutputStream(new
      FileOutputStream("resultats.txt"));

    int nb_alarmes = 0; // nombre d'alarmes désiré
    int t          = 0; // type d'alarme minimal à partir duquel on
      désire être prévenu
    int n          = 0; // niveau d'alarme minimal à partir duquel on
      désire être prévenu
    int t_p        = 0; // type d'alarme minimal prioritaire à partir
      duquel on désiré être prévenu
    int n_p        = 0; // niveau d'alarme minimal prioritaire à
      partir duquel on désire être prévenu


    // Vérification de la conformité des arguments

    String erreur = "Usage : java Generateur <nombre d'alarmes
      desire> (1-1000), <type d'alarme> (1-15), <niveau d'alarme> (1-
      10), <type d'alarme prioritaire> (1-15), <niveau d'alarme
      prioritaire> (1-10)";

    if (args.length != 5) {
      System.out.println(erreur);
      System.exit(1);
    } // fin de if

    try {nb_alarmes = Integer.parseInt(args[0]);
         t          = Integer.parseInt(args[1]);
         n          = Integer.parseInt(args[2]);
         t_p        = Integer.parseInt(args[3]);
         n_p        = Integer.parseInt(args[4]);
    } // fin de try
```

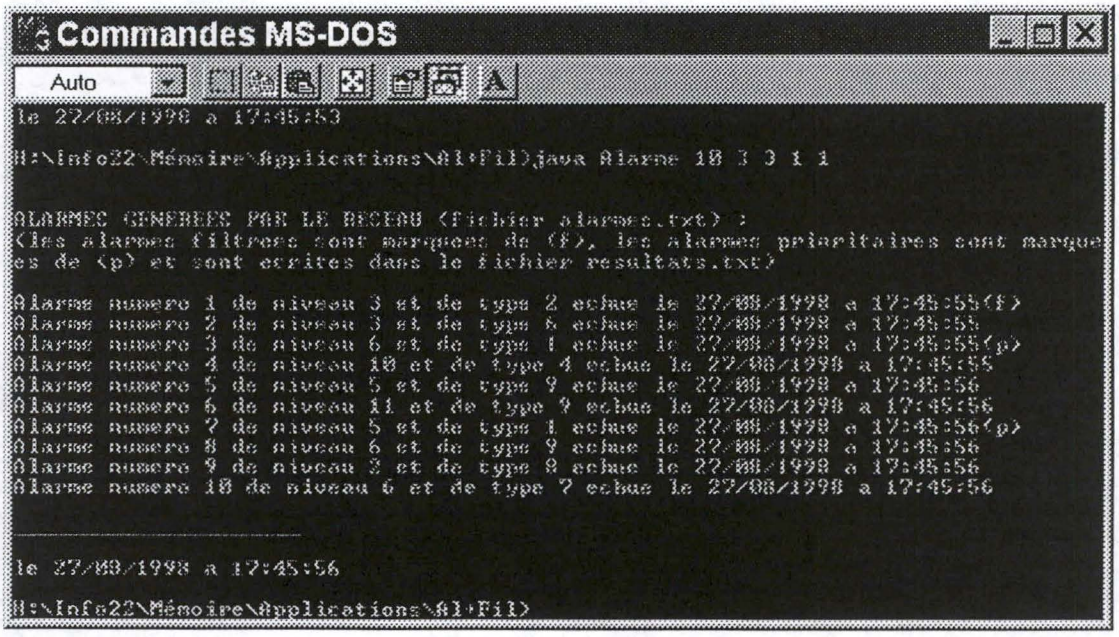
```
    catch (NumberFormatException e) {
      System.out.println(erreur);
      System.exit(1);
    } // fin de catch

    if ((nb_alarmes<1) | (nb_alarmes>1000) |
        (n          <1) | (n          >10)    |
        (t          <1) | (t          >15))   {
      System.out.println(erreur);
      System.exit(1);
    } // fin de if

    System.out.println("");
    System.out.println("");
    System.out.println("ALARMES GENEREES PAR LE RESEAU (fichier
      alarmes.txt) : ");
    System.out.println("(les alarmes correspondant au monitoring sont
      marquees de (*) et sont ecrites dans le fichier
      resultats.txt)");
    System.out.println("");

    for (int i = 1; i<=nb_alarmes; i++) {       // boucle principale
      alarme.générerAlarme();
      alarme.filtrerAlarme(t, n, t_p, n_p);
      alarme.afficherAlarme(i, alarmes, resultats);
    } // fin de boucle for

    System.out.println("");
    System.out.println("_____________________");
    System.out.println("");
    Date dateCourante = new Date();
    String dateString = formatDate.format(dateCourante);
    System.out.println(dateString);

  } // fin de main(args)

} // fin de class Alarme
```

4. Captures d'écran

4.1. Le générateur d'événements

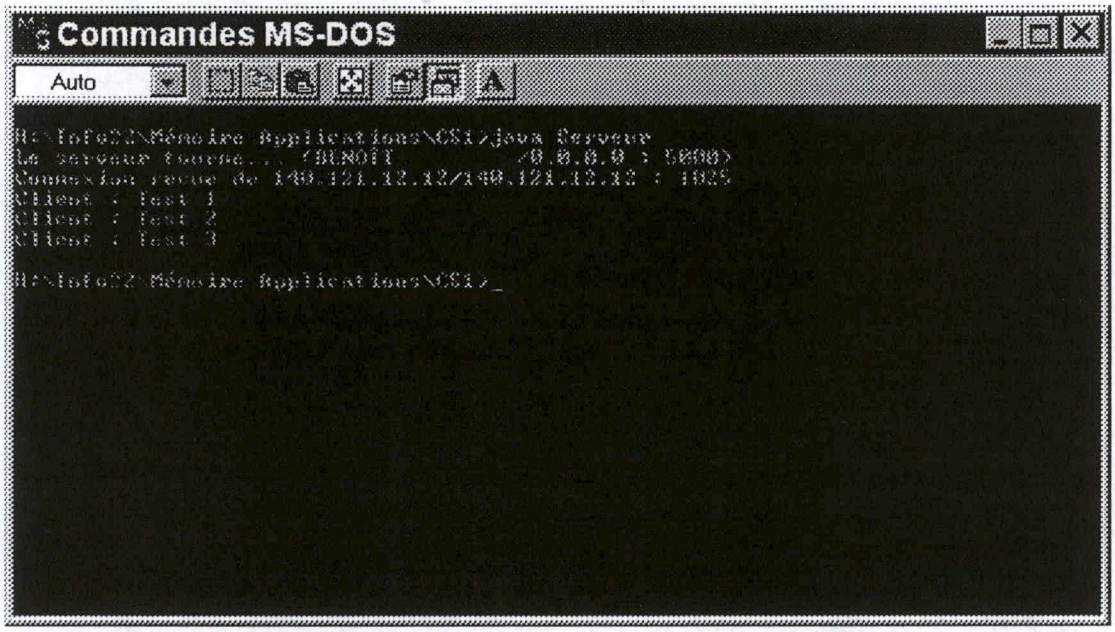


4.2. Le générateur d'événements muni d'un filtre

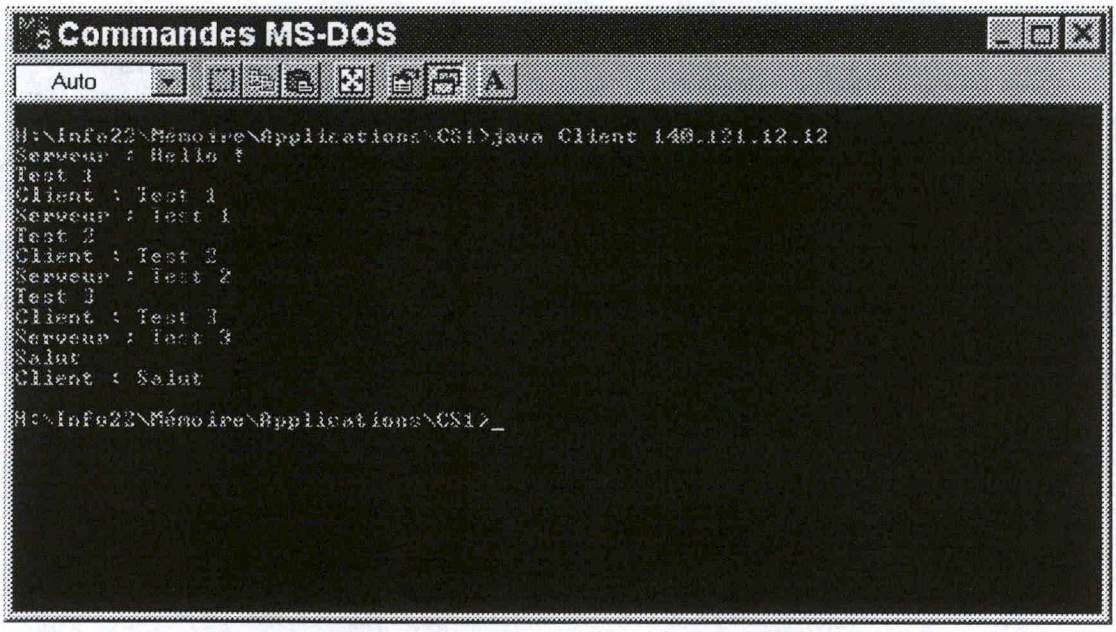


4.3. L'architecture Client/Serveur

4.3.1. Le serveur



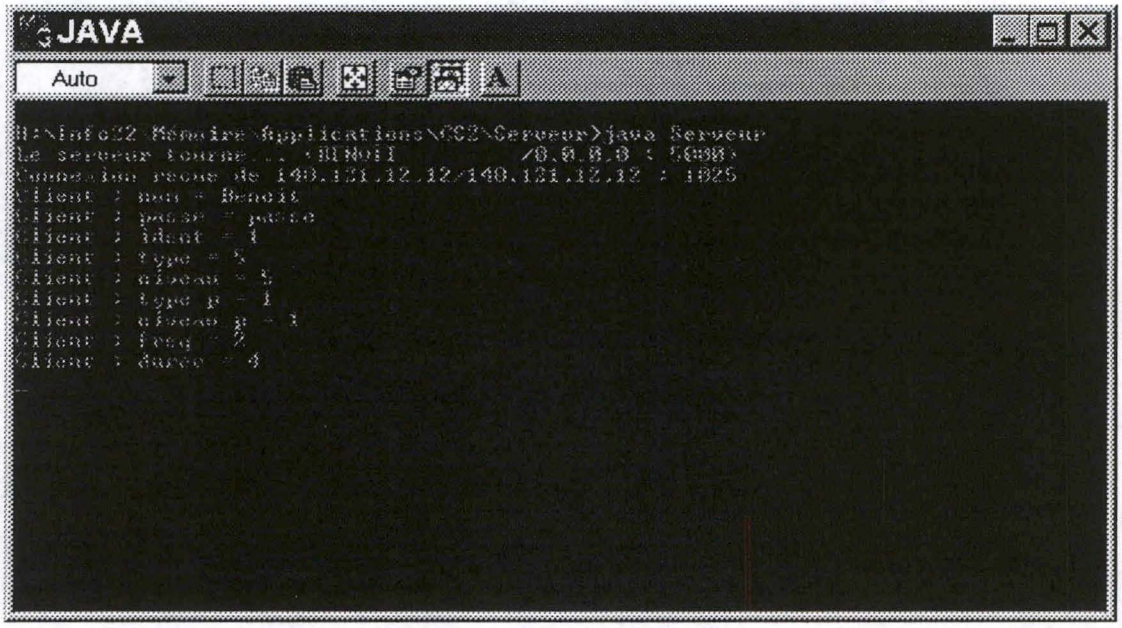
4.3.2. Le client



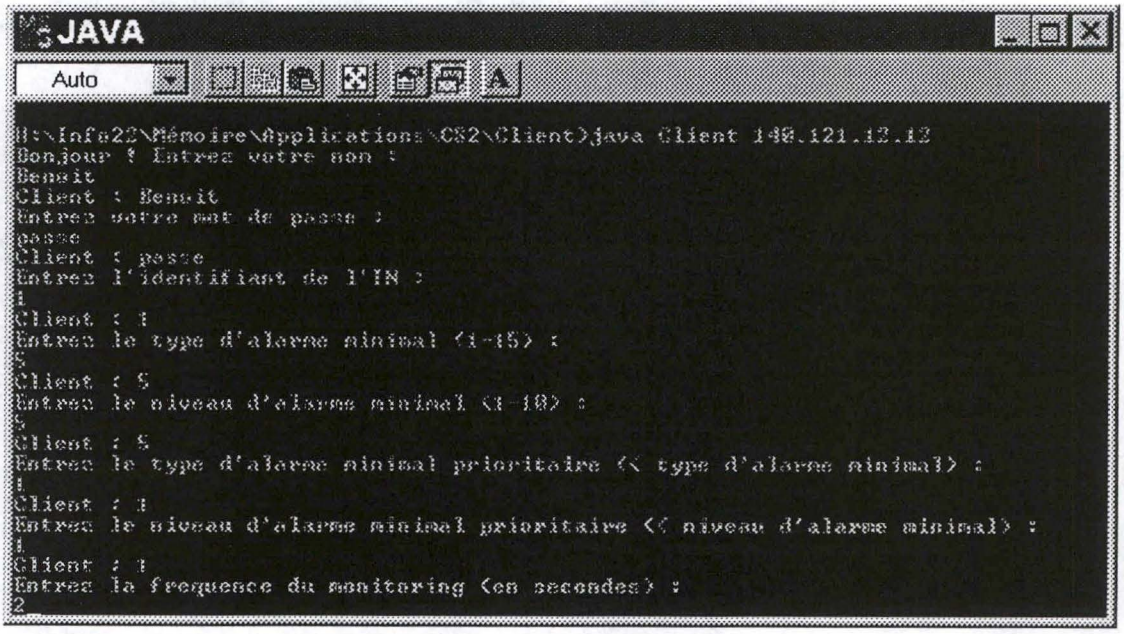
4.4. Le serveur et le client définitifs

4.4.1. La phase d'identification et acquisition des paramètres

4.4.1.1. Le serveur



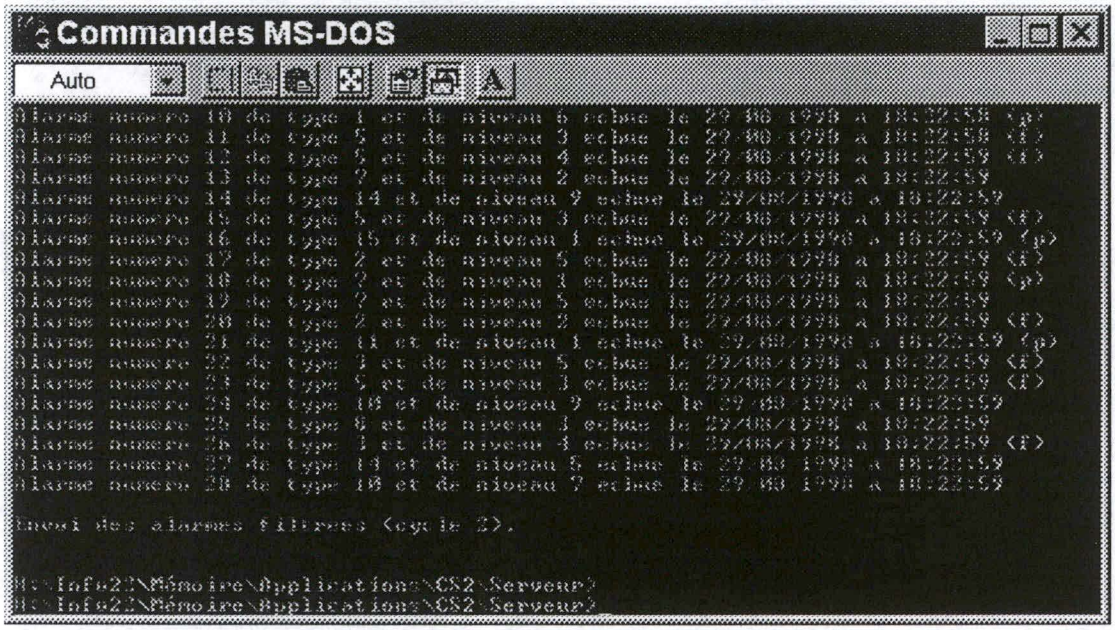
4.4.1.2. Le client[67]



[67] La phase d'identification et d'acquisition des paramètres est, du côté du client, trop longue pour qu'il soit possible d'en afficher toutes les étapes sur un seul écran.

4.4.2. La fin de la phase de monitoring

4.4.2.1. Le serveur



4.4.2.2. Le client